UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RAFAEL SANTOS BEZERRA

Aplicação de Mashups no Gerenciamento de Redes

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof[a]. Dr[a]. Liane M. R. Tarouco
Orientador

Porto Alegre, 20 de Agosto de 2012

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Bezerra, Rafael Santos

Aplicação de Mashups no Gerenciamento de Redes / Rafael Santos Bezerra. – Porto Alegre: PPGC da UFRGS, 2012.

105 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2012. Orientador: Liane M. R. Tarouco.

1. Gerenciamento de redes. 2. Web 2.0. 3. Mashups. I. Tarouco, Liane M. R.. II. Título.

"We will always be more human than we wish to be."
— DANIEL GILDENLÖW

AGRADECIMENTOS

Aos meus pais e irmãos, por tudo.

À minha família e amigos, pelo suporte e, principalmente, pela compreensão necessária para ouvir tantos ”nãos” enquanto eu persegui o objetivo de me tornar mestre.

Aos colegas que vieram comigo do Piauí para o Rio Grande do Sul, pela jornada.

À professora Liane Tarouco, pela valiosa orientação.

Ao professor Lisandro Granville e ao colega Carlos Raniery. Pela muitas horas de discussão. Pelas inúmeras horas dedicadas a revisões. Pelas imensuráveis paciência, dedicação e excelência.

Aos colegas do grupo de redes, pela camaradagem e por serem um grupo com o qual a convivência torna qualquer um imensamente mais inteligente.

Ao CNPQ, pelo apoio financeiro.

À UFRGS, por me receber e por tornar tudo isso possível.

Às madrugadas. Afinal, o que seria de nós sem ela?

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AS	Autonomous System
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
IP	Internet Protocol
P2P	Peer-to-Peer
SOA	Service Oriented Architecture
SOA	Simple Object Access Protocol
RSS	RDF Site Summary/Really Simple Syndication
API	Application Programming Interface
AJAX	Asynchronous JavaScript and XML
HTTP	Hypertext Transfer Protocol
CSV	Comma Separated Values
XML	Extensible Markup Language
MRTG	Multi Router Traffic Grapher
MIB	Management Information Base
MbD	Management by Delegation
TLM	Top Level Manager
MLM	Middle Level Manager
WSDL	Web Services Definition Language
UDDI	Universal Description Discovery and Integration
FTP	File Transfer Protocol
BPEL	Business Process Execution Language
RPC	Remote Procedure Call
DOM	Document Object Model
W3C	World Wide Web Consortium

JSON	JavaScript Object Notation
WS	Web Services
REST	Representational State Transfer
CRUD	Create Read Update Delete
RDF	Resource Description Framework
IDE	Integrated Development Environment
XLS	Excel Spreadsheet
SGBD	Sistema de Gerenciamento de Bancos de Dados
CRM	Consumer Relationship Management
TI	Tecnologia da Informação
SQL	Structured Query Language
RNP	Rede Nacional de Ensino e Pesquisa
URL	Uniform Resource Locator
NMMS	Network Management Mashup System
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
JSP	Java Server Pages
ISP	Internet Service Provider
BGP	Border Gateway Protocol
ASN	Autonomous System Number
BPMN	Business Process Modeling Notation
OID	Object Identifier
PTT	Ponto de Troca de Tráfego
CO	Chefe de Operações
IFC	Information Flow Control
DDoS	Distributed Denial of Service
C&C	Command and Control/Comando e Controle
IRC	Internet Relay Chat
DNS	Domain Name System

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Mashups são aplicações Web criadas a partir da composição de recursos heterogêneos disponíveis na Web. Eles são considerados uma das principais tecnologias da Web 2.0, baseando-se nos princípios de criação de conteúdo pelo usuário final, cooperação e reuso. Esses princípios são aplicados através de sistemas de *mashups*, que consistem em aplicações que permitem que um usuário final realize composições de recursos sem a necessidade de habilidade em desenvolvimento de *software*. A aplicação de tecnologias da Web 2.0, em especial dos *mashups*, no gerenciamento de redes é potencialmente vantajosa, principalmente em cenários onde um administrador deve lidar com múltiplas ferramentas de gerenciamento de forma integrada. Entretanto, não há nenhuma investigação prévia da aplicação dessas tecnologias no gerenciamento. Essa investigação é o trabalho desenvolvido na presente dissertação. Para tal, uma arquitetura para um sistema de *mashups* de gerenciamento de redes é proposta. Com base nessa arquitetura, um protótipo desse sistema é descrito. Esse protótipo é utilizado na criação de *mashups* para dois cenários reais de gerenciamento de redes. A criação desses *mashups* permitiu a avaliação tanto da arquitetura quanto do protótipo. Nessa avaliação, é realizada uma comparação entre a criação dos *mashups* utilizando o protótipo e o desenvolvimento *ad hoc* (*i.e.*, sem o uso de sistemas de *mashups*) dos mesmos. Além disso, é realizada uma avaliação qualitativa dos *mashups* criados, baseada em critérios como facilidade de implementação e utilização, extensibilidade, flexibilidade e confiabilidade. Os resultados das avaliações realizadas indicam que a tecnologia de *mashups* é tanto aplicável quanto vantajosa no contexto de gerenciamento de redes, ainda que a maioria das vantagens dessa aplicação dependam da existência de um sistema de *mashups* que permita que administradores criem suas próprias composições.

Palavras-chave: Gerenciamento de redes, Web 2.0, Mashups.

Application of Mashups in Network Management

ABSTRACT

Mashups are Web applications created through the integration of external resources available on the Web. They have been considered a hallmark of Web 2.0 technologies, allowing end users to develop their own applications and encouraging cooperation and reuse. However, their usage in the network management field remains unexploited. In this context, we look at Web 2.0 as a feasible mechanism able to integrate heterogeneous management information. In this dissertation, we propose an architecture and a system prototype that allows network administrators to design their own management applications through the composition of external resources. The creation of mashups for two network management scenarios allowed us to evaluate both our architecture and our prototype. In this evaluation, we compare the development of network management mashups using our prototype and in an ad hoc fashion. We also perform a qualitative analysis of our mashups, based on characteristics such as ease-of-implementation, ease-of-use, extensibility, flexibility and reliability. The results of our evaluation indicate that mashups are, indeed, both applicable and advantageous in the context of network management, but the advantages of such application largely depend on a mashup system that enables administrators to create their own mashups.

Keywords: Network Management, Mashups, Web 2.0.

1 INTRODUÇÃO

As redes de computadores são entidades em constante crescimento, tanto em tamanho como em complexidade e importância. Esse crescimento, aliado ao fato delas estarem gradualmente tornando-se parte do conjunto básico de serviços de infraestrutura, enfatiza a importância da tarefa de gerenciamento de redes. Tal tarefa visa manter infraestruturas de rede seguras, funcionais e otimizadas. Isso é possível através da ação de administradores de redes especializados, empregando um crescente número de ferramentas e técnicas. Entretanto, apesar do crescimento da quantidade de tecnologias voltadas ao gerenciamento de redes, determinados cenários ainda apresentam desafios aos administradores que empregam essas tecnologias. Um exemplo desse tipo de cenário é a integração de informações advindas de diversas fontes, ou seja, quando um administrador de redes precisa utilizar ferramentas distintas em conjunto para obter indícios do comportamento da rede. Outro importante exemplo é o gerenciamento situacional, que consiste na solução de problemas de natureza imediata que surgem no cotidiano do gerenciamento de redes. Nesses casos, a solução ideal passa pelo uso de uma ferramenta que precisa ser desenvolvida ou adquirida. Dada a natureza do problema, ambos esses processos são, via de regra, custosos em termos de tempo e recursos. Apesar do pouco suporte a esses cenários, espera-se que administradores de rede lidem de forma ágil com um conjunto cada vez maior de informações de diferentes fontes e ferramentas de gerenciamento, as quais raramente são projetadas visando integração ou agilidade no uso em conjunto com outras ferramentas.

A administração de um Sistema Autônomo (*Autonomous System* - AS) é um exemplo de situação de gerenciamento que envolve tanto a questão das múltiplas fontes de dados quanto o gerenciamento situacional. Um administrador de rede, nesse caso, precisa monitorar o tráfego trocado entre o AS gerenciado e seus vizinhos, verificando se o mesmo cumpre Acordos de Nível de Serviço (*Service Level Agreement* - SLA). Para tal, o administrador deve acessar cada um dos roteadores de borda da sua rede, monitorar e analisar o tráfego trocado, além de comparar esse tráfego com SLAs (BATTISTA; REFICE; RIMONDINI, 2006). Cada uma dessas etapas (*i.e.*, recuperação de dados, processamento, visualização da informação, gerenciamento de SLAs e produção de resultados da comparação) envolve uma ou mais ferramentas de gerenciamento de redes (*i.e* fontes de dados) distintas. Além disso, o diagnóstico de problemas e/ou violações de SLA nesses casos nem sempre é um processo direto e previsível. Ele pode demandar informações, visualizações ou processamento dos dados completamente diferentes do normal. Esse tipo de diagnóstico é um exemplo de necessidade situacional de gerenciamento de redes. O método mais empregado atualmente para lidar com situações de gerenciamento como essas é a utilização de *scripts*, pequenos *softwares* domésticos normalmente desenvolvidos pelo próprio administrador. Entretanto, um administrador de redes nem sempre será suficien-

temente proficiente em desenvolvimento de *software* para criar a solução necessária, visto que tal desenvolvimento requer um conjunto de especialidades distinto do gerenciamento de redes. Então, quando a abordagem de *scripting* falha, as alternativas são a aquisição de uma ferramenta comercial que resolva o problema ou o desenvolvimento *in loco* de uma ferramenta de gerenciamento de maior porte adequada, normalmente desempenhado por uma equipe de desenvolvimento de *software*.

Cada uma dessas abordagens envolve custos potencialmente elevados, seja para treinamento de administradores de rede em desenvolvimento de *software*, contratação de desenvolvedores, aquisição de ferramentas comerciais e/ou treinamento de pessoal nas mesmas. Além disso, também há o problema do custo de tempo associado a tais abordagens. Esse tempo é gasto, por exemplo, em desenvolvimento e manutenção de *software*, capacitação de pessoal e/ou processos administrativos relativos a aquisições de ferramentas comerciais. Tais problemas, não raro, tornam tanto o desenvolvimento *in loco* quanto a aquisição de ferramentas comerciais inviáveis para a solução de necessidades de gerenciamento situacionais (MOHAMMADI; KHALILI; ASHOORI, 2009). Isso acontece porque o escopo caracteristicamente limitado dessas aplicações não justifica os custos associados. Além disso, a natureza efêmera de um problema situacional requer uma agilidade maior do que o que geralmente se consegue ao empregar abordagens como *scripting*, desenvolvimento de ferramentas de gerenciamento domesticas ou aquisição de ferramentas comerciais. Finalmente, tais abordagens podem incorrer em imprecisões detrimentais para o administrador de redes, seja por limitações de tal administrador quando desempenhando o papel de desenvolvedor de *software*, por detalhes do problema perdidos na comunicação entre administrador e desenvolvedor ou por ferramentas comerciais não cobrirem detalhes intrínsecos ao problema da rede gerenciada, visto que essas normalmente abordam classes de problemas, não problemas específicos.

O *framework* SNMP (Simple Network Management Protocol), atualmente a solução *de facto* para gerenciamento de redes TCP/IP, não aborda diretamente os problemas caracterizados. Sua maior contribuição nesse sentido é prover tanto um protocolo quanto um formato de dados padronizados para recuperar informações e realizar operações em diversos elementos gerenciados distintos (HARRINGTON; PRESUHN; WIJNEN, 2002). Num cenário envolvendo integração de múltiplos recursos ou gerenciamento situacional, o SNMP torna-se mais um recurso, o qual, normalmente, gera grandes quantidades de dados. A integração e a visualização desses dados com outras fontes não faz parte do escopo do *framework*. Esforços nesse sentido existem (SCHONWALDER et al., 2007), mas esses envolvem extenso emprego de *scripting* e técnicas de desenvolvimento de *software*. Graças a essas e outras limitações, o SNMP é considerado insuficiente como solução única para o gerenciamento de redes atual (SOLDATOS; ALEXOPOULOS, 2007). Isso tem motivado a academia a pesquisar a aplicação de tecnologias bem sucedidas em outras áreas no gerenciamento de redes. Essa tendência pode ser exemplificada pelos estudos da aplicação de *peer-to-peer* (P2P) (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004), Computação Autonômica (NOBRE; GRANVILLE, 2010), Arquiteturas Orientadas a Serviço (SOA - *Service Oriented Architecture*) (VIANNA et al., 2007) e Web Services (CURBERA et al., 2002). Dentre essas iniciativas, a mais importante para o contexto do presente trabalho é a aplicação de tecnologias relacionadas a Web Services e SOA (LIU et al., 2007). Sob esse paradigma, as aplicações são serviços independentes de plataforma que podem ser utilizados internamente e/ou por parceiros, sendo possível também a sua composição (TSAI, 2005). No caso do gerenciamento, ferramentas sofisticadas podem ser criadas a partir dessa composição, empregando serviços de ge-

renciamento mais simples, como Web Services de acesso a elementos gerenciados via SNMP. Entretanto, tecnologias da SOA, como o SOAP (*Simple Object Access Protocol*), são voltadas a desenvolvedores de *software*, sendo sua assimilação complexa. Isso as tornas inadequadas para administradores de rede, os quais, como já abordado, não são especialistas em desenvolvimento. Além disso, o desenvolvimento utilizando SOA ainda é um processo razoavelmente demorado, não sendo adequado para aplicações situacionais (MAXIMILIEN; RANABAHU; TAI, 2007). Uma tecnologia da Web 2.0, denominada de *Mashups*, apresenta-se como alternativa para resolver os problemas levantados, aliando a modularização e reusabilidade do SOA com princípios de agilidade e foco no usuário final (O'REILLY, 2005).

Os *mashups* são o foco de estudo desse trabalho. Eles consistem em aplicações criadas a partir da integração de diversas fontes de dados e funcionalidade disponíveis na rede (MERRIL, 2003), tais como *feeds* RSS, Web Services, APIs *online*, dentre outros. De acordo com as principais propostas, espera-se que a criação desses *mashups* seja realizada pelo usuário final de forma ágil e dinâmica. Entretanto, a atividade de integrar diversas fontes heterogêneas de dados e aplicações é uma tarefa complexa, que envolve um conhecimento em técnicas de desenvolvimento de *software* que não pode ser esperado do usuário final. Esse desenvolvimento, portanto, deve ser possibilitado por ferramentas de criação de *mashups*, ou sistemas de *mashup* (*Mashup Systems*). Para tal fim, essas ferramentas, normalmente sistemas Web, empregam tecnologias voltadas à usabilidade, como AJAX e Flash, para apresentar uma interface amigável e altamente interativa ao usuário, abstraindo detalhes técnicos da integração sob semânticas de alto nível. Tais detalhes técnicos, automatizados pelo próprio sistema de *mashups*, incluem a interação entre protocolos diferentes (*e.g.*, SOAP, HTTP), diferentes formatos de dados (*e.g.*, CSV, XML), APIs *online* (*e.g.*, Twitter API, Google Maps API), dentre outros. Com o provisionamento de tais facilidades, os sistemas de *mashup* possibilitam também a criação de *mashups* situacionais (MAXIMILIEN; RANABAHU; TAI, 2007). Estes últimos consistem em *softwares* de pequeno porte, modulares e reutilizáveis cujo desenvolvimento utilizando métodos tradicionais seria demasiadamente custoso em termos de tempo e esforço. O propósito dessas aplicações situacionais é atender necessidades de natureza imediata, resolvendo problemas de forma ágil e sendo descartáveis sem que isso apresente prejuízo ou sendo reusáveis sem que esse reuso requeira esforço demasiado. Exemplos de sistemas de *mashups* voltados a criar esse tipo de aplicação são o Yahoo! Pipes e o JackBe Presto.

A aplicação de *mashups* no gerenciamento de redes visa trazer à área uma tecnologia de integração de recursos que aproxime o administrador de rede da tarefa de desenvolvimento de ferramentas de gerenciamento. Através do uso de um sistema de *mashups*, um administrador pode criar um *mashup* de gerenciamento de forma ágil e simplificada, adequada a um usuário inexperiente em desenvolvimento de *software*. Assim, espera-se fornecer ao administrador a capacidade de integrar informações e funcionalidades de diversas fontes heterogêneas. Tal capacidade é um recurso valioso no gerenciamento de redes (SOLDATOS; ALEXOPOULOS, 2007), tendo em vista o problema das múltiplas fontes abordado e, também, o crescente número de fontes de dados de gerenciamento que expõem seus dados em formatos Web (*e.g.*, Netflow, MRTG), os quais *sistemas de mashup* se propõem a suportar. O emprego de um sistema *mashups* de gerenciamento de redes também aborda diretamente questões do gerenciamento situacional. De acordo com a proposta dos *mashups* (MOHAMMADI; KHALILI; ASHOORI, 2009), um *mashup* de gerenciamento pode ser utilizado como uma ferramenta situacional adequada especificamente ao problema imediato do administrador. Além disso, questões de imprecisão rela-

tivas a esses cenários são mitigadas, visto que, ao possibilitar que o próprio administrador crie seus *mashups* de gerenciamento, evita-se que detalhes do problema a ser resolvido sejam perdidos numa comunicação entre administrador de redes e desenvolvedor de *software*. Outra possibilidade trazida pelos *mashups* ao gerenciamento é o compartilhamento dessas aplicações. Ao permitir que um administrador reutilize um *mashup* criado por outro administrador, propicia-se a cooperação entre administradores de redes através do reuso. Finalmente, a Internet está rapidamente se movendo de uma rede de dispositivos para uma rede de serviços, mudança a qual o gerenciamento de redes deve acompanhar (SCHONWALDER et al., 2009). As características dos *mashups* são um passo nesse sentido, trazendo a integração de vários serviços, agilidade e foco nos problemas de gerenciamento em si, bem como no administrador e nas suas necessidades.

O objetivo dessa pesquisa é, portanto, propor uma solução de gerenciamento de redes que possibilite a um administrador de redes lidar com diversos recursos distintos e problemas de gerenciamento situacionais, através da criação de suas próprias ferramentas de gerência. Pelas razões abordadas, a tecnologia de *mashups* foi escolhida para essa solução, materializada em um sistema de *mashups* de gerenciamento de redes. Para investigar a aplicação dos *mashups* no gerenciamento, foi conduzido um estudo sobre os principais sistemas de *mashup open-source* disponíveis na Internet. Paralelamente, foram desenvolvidos estudos de caso utilizando *mashups* desenvolvidos de forma *ad hoc* (*i.e.*, sem o uso de um sistema de *mashups*) para a solução de problemas de gerenciamento envolvendo integração de recursos heterogêneos. Baseados no conhecimento adquirido nessas atividades, foi desenvolvida uma arquitetura de referência para um sistema de *mashups* de gerenciamento de redes. Tal arquitetura é baseada no modelo de três camadas proposto por (ECKERSON, 1995) para sistemas Web e agrega as principais características observadas nos sistemas de *mashup* estudados que podem ser aplicadas ao gerenciamento. Com base nessa arquitetura de referência, um protótipo de sistema de *mashups* foi desenvolvido. O protótipo é um sistema Web que permite a criação de *mashups* de gerenciamento através de uma interface *drag-and-drop*. Esse sistema foi utilizado para desenvolver *mashups* para estudos de caso levantados. Para avaliar o protótipo, seu uso no desenvolvimento dos *mashups* dos estudos de caso é comparado com o processo de desenvolvimento *ad hoc*. Os *mashups* criados, por sua vez, são avaliados qualitativamente com base em critérios como esforço de implementação e utilização, extensibilidade, flexibilidade e confiabilidade.

Essa dissertação está organizada da seguinte forma: O Capítulo 2 trata da base teórica do trabalho, abordando um panorama do gerenciamento de redes, bem como detalhes sobre a Web 2.0, tratando também em profundidade sobre a tecnologia de *mashups* e sua aplicação no gerenciamento. Em seguida, a arquitetura de referência para um sistema de *mashups* de gerenciamento de redes desenvolvida é apresentada, no Capítulo 3. O Capítulo 4 descreve o protótipo de sistema de *mashups* desenvolvido. No Capítulo 5, são discutidos os estudos de caso de gerenciamento abordados, bem como é realizada a avaliação do protótipo e dos *mashups* de gerenciamento criados. O capítulo 6 apresenta as conclusões e sugestões dos próximos passos a serem seguidos na investigação da aplicação de *mashups* no gerenciamento de redes.

2 GERENCIAMENTO DE REDES, WEB 2.0 E MASHUPS

O presente capítulo apresenta o embasamento teórico deste trabalho. A primeira seção faz uma breve revisão dos principais trabalhos na área de Gerenciamento de Redes, abordando também a evolução desses e os problemas atuais do gerenciamento. A seção seguinte descreve o panorama da Web 2.0 e suas principais tecnologias. A seguir, uma seção é dedicada a abordar em profundidade os *Mashups*, principal objeto de estudo deste trabalho. Finalmente, a última seção do capítulo trata da aplicação dos *mashups* no gerenciamento de redes.

2.1 Contexto atual do gerenciamento de redes

O *framework* SNMP (*Simple Network Management Protocol*) foi definido na década de 80 com o objetivo de ser uma solução para o gerenciamento de redes TCP/IP (HARRINGTON; PRESUHN; WIJNEN, 2002). Esse *framework* estabelece um protocolo de comunicação entre gerente (*i.e.*, a estação de gerenciamento) e agente (*i.e.*, elementos gerenciados como, por exemplo, equipamentos de rede). Através desse protocolo, o gerente pode tanto recuperar informações sobre os elementos gerenciados quanto manipular parâmetros de configuração dos mesmos. Tais informações e parâmetros são codificados em uma base hierárquica de dados denominada *Management Information Base* (MIB). Finalmente, o *framework* define um método de comunicação que permite que agentes enviem notificações assíncronas para gerentes na forma de *traps*. O mecanismo de *traps* permite, por *exemplo*, que o agente SNMP de um roteador emita um aviso para a estação de gerenciamento caso seja detectada uma falha em algum dos seus enlaces de comunicação.

Historicamente, o *framework* SNMP foi usado seguindo um paradigma centralizado, ou seja, onde há uma única estação de gerenciamento central responsável por administrar os vários agentes da rede. O gerenciamento centralizado utilizando SNMP foi, por muito tempo, uma solução suficiente para o gerenciamento de redes TCP/IP, tornando-se o padrão *de facto* para o gerenciamento desse tipo de rede (LEINWAND; CONDROY, 1996). Essa centralização, no entanto, não é uma exigência do *framework*, o qual suporta paradigmas distribuídos (*i.e.*, com múltiplos gerentes). Contudo, o crescimento das redes de computadores e da Internet nas duas últimas décadas expôs, gradualmente, limitações do paradigma de gerenciamento centralizado. Por exemplo, a escalabilidade do paradigma é limitada pela capacidade da estação central, que pode ser insuficiente para lidar com a elevada quantidade de operações de gerenciamento de uma rede de grande porte. Além disso, em caso de falha nessa estação, todo o gerenciamento da rede é comprometido, o que é uma severa limitação de tolerância a falhas.

Devido às limitações do paradigma centralizado, surgiram novas soluções para o gerenciamento de redes, incluindo soluções baseadas em paradigmas de gerenciamento dis-

tribuídos e cooperativos (SCHONWALDER; QUITTEK; KAPPLER, 2000), onde vários gerentes atuam no gerenciamento de uma mesma rede. Esses novos paradigmas apresentam uma melhor escalabilidade, visto que eles possibilitam que a complexidade do gerenciamento de uma rede de grande porte seja distribuída entre diversas estações de gerenciamento. Essa distribuição também elimina a existência de um ponto central de falha na gerência da rede, pois, em caso de falha, outras estações podem assumir as responsabilidades da estação de gerenciamento comprometida.

Um exemplo de solução de gerenciamento baseada no paradigma distribuído é o gerenciamento por delegação (*Management by Delegation* - MbD) (GOLDSZMIDT; YEMINI, 1995), onde os vários gerentes são organizados hierarquicamente e gerentes de alto nível (*Top Level Manager* - TLM) podem delegar funções a gerentes intermediários (*Middle Level Manager* - MLM). Uma importante implementação do MbD é o gerenciamento baseado em *scripts* através da Script MIB (SCHONWALDER; QUITTEK; KAPPLER, 2000). Ela consiste em um módulo da MIB projetado para implementação em MLMs e seu objetivo é permitir que, através do protocolo SNMP, TLMs deleguem tarefas complexas a esses MLMs definidas em *scripts* de gerenciamento criados por administradores. Os MLMs, por sua vez, são responsáveis pela execução dos *scripts* que lhe são delegados e por suportar os ambientes de execução necessários para tal.

Tal como aconteceu com o paradigma centralizado, o crescimento das redes também expôs limitações do *framework* SNMP (SOLDATOS; ALEXOPOULOS, 2007). Por exemplo, como o SNMP foi definido visando o gerenciamento de dispositivos (*e.g.,* roteadores e *switches*), ele é pouco utilizado no gerenciamento de serviços disponíveis na rede (*e.g.,* contas e servidores de email, servidores de aplicação, ambientes virtualizados), ainda que haja algum suporte para isso disponível *online* (*e.g.,* o módulo da MIB Mod-Apache-SNMP [1]). Essa característica tem se tornado uma limitação nos últimos anos, pois as redes atuais têm evoluído de redes de dispositivos para redes de serviços (SCHONWALDER et al., 2009).

Em resposta às limitações do SNMP e às necessidades das redes atuais, um comportamento recente da comunidade de pesquisa consiste em buscar tecnologias consolidadas em outras áreas e aplicá-las no gerenciamento (BEZERRA et al., 2010). Por exemplo, a aplicação de Web Services no gerenciamento de redes trouxe resultados positivos nos últimos anos, tais como a mitigação de problemas relacionados à travessia de *firewalls* comumente enfrentados por tráfego SNMP, empregando para tal protocolos de aplicação mais bem aceitos como o HTTP (*Hypertext Transfer Protocol*) (VIANNA et al., 2007). O objetivo da aplicação dessa tecnologia é oferecer serviços modulares, independentes e disponíveis *online*, denominados *Web Services*, os quais vão realizar funções de gerenciamento. A interação com esses serviços é realizada através de um protocolo padronizado (*e.g.,* SOAP - *Simple Object Access Protocol*) e eles são descritos em um formato também padronizado (*e.g.,* WSDL - *Web Services Definition Language*). Esses serviços podem ser publicados e descobertos através de um registro centralizado (*e.g.,* UDDI - *Universal Description Discovery and Integration*). Além dessa padronização, outra vantagem dos Web Services é que eles são suportados por um amplo ferramental disponível na Internet, que inclui, por exemplo, bibliotecas que facilitam a utilização do protocolo SOAP e *softwares* para a geração automática de WSDLs.

A aplicação de novas tecnologias no gerenciamento de redes e o constante surgimento de novas ferramentas culminaram em um cenário problemático para o administrador de redes. O problema consiste na grande quantidade de ferramentas de gerenciamento com

[1] http://mod-apache-snmp.sourceforge.net/english/docs.htm

as quais esse administrador de redes deve lidar no seu cotidiano (*i.e.,* o problema das múltiplas ferramentas). Esse problema é agravado pelo fato de que, geralmente, essas ferramentas não são projetadas levando em consideração que o administrador pode precisar integrá-las (DOS SANTOS et al., 2010). Isso acontece porque a maioria das ferramentas de gerenciamento atuais são soluções fim-a-fim para problemas específicos ou para uma classe de problemas, um exemplo disso é a ferramenta *Multi Router Traffic Grapher* (MRTG) (OETIKER, 2008), voltada à criação de gráficos informativos sobre a rede em tempo real. Outro agravante para o problema das múltiplas ferramentas pode ser visto no gerenciamento de dispositivos de rede, onde fabricantes de equipamentos têm implementado o acesso a funcionalidades importantes dos seus produtos através de métodos de acesso proprietários (*e.g.,* ferramentas próprias, módulos proprietários da MIB e/ou protocolos proprietários), ainda que operações mais básicas (*e.g.*, manipulação de tabelas de roteamento) sejam possíveis via SNMP. Isso força o administrador de redes a lidar também com ferramentas específicas para determinados fabricantes.

Administradores de rede normalmente recorrem à criação de *scripts* para lidar com situações que envolvem múltiplas ferramentas de gerenciamento, empregando, para tal, linguagens como o Bash. Esses *scripts* consistem em pequenos *softwares* que invocam múltiplas ferramentas seguindo uma lógica específica para atender uma determinada necessidade. Por exemplo, um *script* pode invocar um cliente SNMP para recuperar dados de diversos equipamentos da rede, submeter esses dados a uma ferramenta de geração de relatórios de inventário e salvar o relatório em uma determinada localidade na rede usando FTP (*File Transfer Protocol*).

O principal problema da criação de *scripts* pelo administrador de redes é que o gerenciamento de redes e o desenvolvimento de *software* são disciplinas diferentes com conjuntos de conhecimentos específicos distintos (DOS SANTOS et al., 2010). Além disso, a integração de múltiplas ferramentas com dados de fontes heterogêneas é um problema de desenvolvimento de *software* complexo (TUCHINDA; SZEKELY; KNOBLOCK, 2008). Então, ainda esse administrador tenha alguma habilidade de desenvolvimento, é provável que ele encontre diversos cenários de integração de recursos cuja complexidade está além das suas capacidades. Finalmente, a criação de *scripts*, como qualquer processo de desenvolvimento de *software*, tem um custo de tempo normalmente elevado. Esse custo pode ser impeditivo para um administrador de redes, visto que a responsabilidade do mesmo não é criar *scripts* de gerenciamento, mas sim manter o bom funcionamento da rede.

Quando, por questões de tempo disponível ou habilidade em desenvolvimento, o administrador de redes não é capaz de criar um *script* necessário para atender uma determinada necessidade de gerenciamento, sua principal alternativa é obter uma nova ferramenta que atenda essa necessidade. As duas principais abordagens empregadas nesse caso são: recorrer a desenvolvedores de *software* para criar uma nova ferramenta ou adquirir uma ferramenta comercial. Um dos principais problemas de ambas as abordagens é que elas podem resultar em soluções que não levam em consideração requisitos importantes da necessidade tratada. No caso da primeira abordagem, desenvolvedores de *software* dificilmente têm a experiência e o conhecimento em gerenciamento de redes necessários para compreender completamente as necessidades de um administrador de redes. Ferramentas comerciais, por sua vez, são projetadas visando resolver uma classe de problemas relacionada a um cenário de gerenciamento, não voltadas a atender necessidades específicas de um determinado administrador, então é provável que detalhes únicos à necessidade desse administrador (*e.g.*, questões relacionadas à topologia da rede gerenciada) não sejam suportados por tais ferramentas. Nesse caso, muitas vezes os administradores solicitam que

desenvolvedores de *software* implementem customizações nessas ferramentas comerciais, solução que pode apresentar os mesmos problemas da primeira abordagem. Outro problema a ser considerado é que os custos de tempo e recursos financeiros associados a ambas as abordagens descritas podem torná-las inviáveis, caso a necessidade atendida por elas seja de natureza efêmera devido a, por exemplo, alta urgência ou escopo muito específico (*i.e.*, necessidades situacionais) (MOHAMMADI; KHALILI; ASHOORI, 2009). Finalmente, essas abordagens têm como resultado uma nova ferramenta, seja esta desenvolvida localmente ou adquirida. Ainda que essa nova ferramenta satisfaça uma necessidade específica do administrador de redes, ela irá contribuir para agravar o problema das múltiplas ferramentas com as quais esse administrador deve lidar.

Uma possível solução para o problema das múltiplas ferramentas no gerenciamento de redes consiste na ideia de aproximar o administrador de redes do desenvolvimento de ferramentas de gerenciamento, através do emprego de tecnologias que possibilitem a integração de recursos (*e.g.*, dados, ferramentas de gerenciamento) e eliminem a necessidade do emprego de *scripts*. A composição de Web Services aplicada ao gerenciamento de redes segue esse princípio (VIANNA et al., 2007), entretanto as tecnologias envolvidas (*e.g., Business Process Execution Language* - BPEL) são orientadas a desenvolvedores de *software*, demandando habilidades em desenvolvimento que não podem ser exigidas de administradores de redes. Tecnologias da Web 2.0, por outro lado, têm como um de seus objetivos eliminar a necessidade dessas habilidades, sendo orientadas ao usuário final, conceitualmente leigo em desenvolvimento de *software* (O'REILLY, 2005). Uma dessas tecnologias, denominada *mashups* (MERRIL, 2003), tem como objetivo possibilitar que um usuário final (*e.g.*, um administrador de redes) crie novas aplicações através da composição de recursos heterogêneos disponíveis na Web, os quais que podem incluir, por exemplo, ferramentas de gerenciamento. As tecnologias da Web 2.0 e os *mashups* serão discutidos em detalhes nas próximas subseções.

2.2 Web 2.0

O termo Web 2.0 (O'REILLY, 2005) define um conjunto de novas aplicações Web baseadas em princípios de interatividade e cooperação. Como pode ser visto pela figura 2.1, nessas novas aplicações o usuário pode e é encorajado a gerar conteúdo, exercendo o papel proativo de colaborador. Essa é a principal mudança em relação a aplicações pré-Web 2.0 (*i.e.*, Web 1.0), onde o usuário exerce o papel passivo de consumidor de conteúdo criado e publicado por grandes empresas de *mídia*. Ao permitir e encorajar a criação de conteúdo pelo usuário final, a Web 2.0 traz consigo a vantagem de uma Internet mais democrática, na qual o controle do conteúdo disponível está cada vez mais difundido entre os usuários (MURUGESAN, 2007). Essa democratização permite, por exemplo, que usuários criem e compartilhem conteúdo relacionado a interesses altamente específicos que, por atingirem um público restrito, recebiam pouca atenção dos grandes *sites* e portais da Web 1.0.

A Wikipedia é um exemplo de aplicação que evidencia as mudanças trazidas pela Web 2.0 representadas na figura 2.1. Sua proposta é ser uma enciclopédia cooperativa onde o usuário possui diversos perfis: autor, revisor e leitor. Uma prova da eficiência desse modelo é um estudo comparativo entre a *Wikipedia* e a tradicional *Enciclopédia Britannica*, o qual demonstrou que a primeira tem uma média de 4 erros por artigo, contra 3 da segunda (NATURE, 2005). Além disso, a Wikipedia supera a *Britannica* em número de artigos, a primeira contendo aproximadamente 3.5 milhões de artigos em língua inglesa, contra

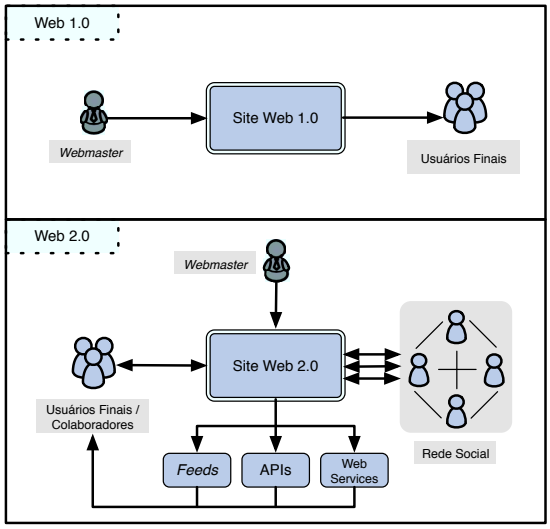

Figura 2.1: Aplicações da Web 1.0 vs Aplicações da Web 2.0

aproximadamente 65 mil da segunda. Nesse caso, as contribuições dos usuários permitiram à enciclopédia livre demonstrar vantagens em relação a uma enciclopédia tradicional como a *Britannica*.

Além da produção, a classificação de conteúdo pelos usuários através, por exemplo, de metadados como *tags*, também é uma forma de aproveitar o esforço coletivo. Essa classificação traz para o usuário informações sobre o contexto no qual determinado conteúdo se insere, além de permitir a automatização de processamento semântico desse conteúdo por *software*. Um exemplo de sucesso da classificação de conteúdo pode ser observado no *site* de compras *online Amazon*[1]. Nele, usuários podem marcar produtos com palavras chave (*i.e.*, tags), as quais podem ser processadas pelo *site*. Através desse processamento, o *site* disponibiliza aos seus usuários funcionalidades como a busca por produtos similares, organização de produtos por *tags*, busca por palavras chave e recomendações baseadas em *tags* previamente empregadas. Além disso, a marcação de produtos com *tags* permite que os administradores da Amazon compreendam melhor como os usuários do *site* enxergam os produtos disponibilizados. É possível observar que a colaboração é um elemento chave tanto no caso da Wikipedia quanto da Amazon. A partir de tal colaboração surge o conceito de 'arquitetura de participação' (AMER-YAHIA; HALEVY, 2007), onde cada usuário agrega valor a um *site* ou aplicação através da criação de conteúdo e/ou da classificação e organização de conteúdo já existente.

A Web 2.0 se beneficiou de mudanças significativas no desenvolvimento de *software* que ocorreram nas últimas décadas. Essas mudanças envolvem a substituição de sistemas monolíticos, altamente acoplados e centralizados, por sistemas distribuídos, altamente reutilizáveis e para os quais a Web funciona como plataforma. Esse novo modelo, onde a Web 2.0 serve como plataforma de aplicações, é um dos principais conceitos propostos por Tim O'Reilly (O'REILLY, 2005). A Web 2.0 também se beneficiou de mudanças no ciclo de vida do *software*. Em contraste com o modelo tradicional de entrega de um produto completo e o subseqüente lançamento de pacotes de atualização durante a manutenção, aplicações da Web 2.0 normalmente são sistemas em constante desenvolvimento. Desde o lançamento, tais sistemas passam por constantes alterações e melhorias base-

[1] www.amazon.com

adas no *feedback* dos usuários, visando atender necessidades cada vez mais específicas dos mesmos. Dessa forma, o usuário desempenha o papel de testador da aplicação, colaborando para a constante melhoria da mesma. Um exemplo do emprego com sucesso desse tipo de ciclo de vida do *software* é o Gmail[1], que esteve em estado beta por 5 anos e, ao finalizar oficialmente essa etapa em 2009, já era um dos provedores de email mais utilizados no mundo.

Visto que o usuário final exerce um papel central na Web 2.0, é natural que as aplicações tenham de atender altos requisitos de usabilidade (O'REILLY, 2005). Tal usabilidade é obtida através do emprego de tecnologias como AJAX (*Asynchronous JavaScript and XML*) (GARRET, 2005) e *Ruby On Rails* (THOMAS, 2008). Essas tecnologias permitem a criação de aplicações Web com interfaces gráficas mais próximas de aplicações *desktop*. As aplicações também são projetadas para permitir que usuários desempenhem tarefas de geração conteúdo e interação com outros usuários com facilidade. A capacidade de realizar tais tarefas em ambientes de alta usabilidade e altamente interativos é o que vai atrair o usuário final a um *site*. Atrair e manter uma base de usuários é de vital importância em uma aplicação da Web 2.0, visto que o valor desse tipo de aplicação está nas contribuições e interações desses usuários (MURUGESAN, 2007).

Tais características de alta usabilidade, interatividade e foco no usuário final que compõem a Web 2.0 são possibilitadas por diversas tecnologias que surgiram e popularizaram-se ao longo da última década. Nas subseções a seguir, são apresentadas as principais dentre essa gama de tecnologias, com foco naquelas que têm um papel central na criação de *mashups*, objeto de estudo desse trabalho.

2.2.1 AJAX

Abreviação para *Asynchronous JavaScript and XML*, o modelo AJAX define um grupo de tecnologias utilizadas em conjunto para permitir que uma aplicação Web realize múltiplas chamadas ao servidor sem a necessidade de uma recarga de página a cada requisição. Ao permitir esse tipo de comunicação mais complexa com o servidor, o AJAX possibilita um aumento na responsividade das aplicações Web, aproximando seu modelo de interação com o usuário ao de uma aplicação *desktop* (THOMAS, 2008). Esse aumento de responsividade pode ser observado, por exemplo, em uma aplicação contendo um formulário a ser preenchido pelo usuário e validado pelo servidor. Pelo modelo Web convencional, o usuário preenche o formulário e o submete ao servidor. Esse servidor, por sua, vez realiza a validação dos campos do formulário e responde para o usuário se os dados preenchidos são válidos. Em caso de dados inválidos, o usuário deve realizar o preenchimento novamente e re-submeter o formulário. Em contraste, essa mesma aplicação, seguindo o modelo AJAX, pode realizar uma validação granular, realizando chamadas ao servidor a cada campo preenchido e informando para o usuário, em tempo real, se o formulário está sendo preenchido corretamente. Nesse caso, é possível perceber que o uso de AJAX não somente agiliza a interação do usuário com a aplicação exemplo, ao evitar repetidos preenchimentos de formulário, como permite que o usuário receba mais *feedback* da aplicação durante a interação.

Como pode ser visto na figura 2.2, um sistema Web baseado em AJAX pode realizar chamadas ao servidor de forma assíncrona, através de requisições em *background* utilizando uma *engine* AJAX. Essa *engine*, executada no navegador Web do cliente, utiliza o objeto *XMLHttpRequest*, normalmente disponibilizado por bibliotecas da linguagem

[1] mail.google.com

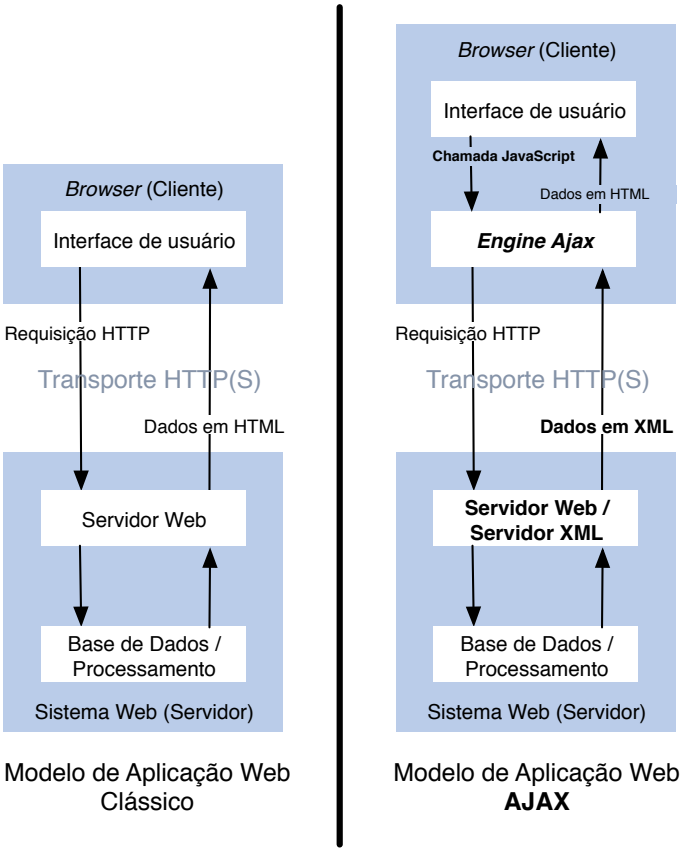

Figura 2.2: Arquitetura Clássica de Aplicações Web vs. Arquitetura AJAX.

JavaScript, para trocar dados com o servidor através de requisições HTTP. As requisições e respostas HTTP contém, no seu campo de dados, dados codificados em XML (*Extensible Markup Language*) representando a comunicação entre o navegador do cliente e o *backend* do *sistema*. Para permitir que o navegador Web faça requisições mais sofisticadas ao servidor, normalmente são utilizados padrões de chamada de procedimento remoto (*Remote Procedure Call* - RPC) baseados em XML, como o XML-RPC (WINER, 1999). Os dados nessas mensagens são codificados e manipulados utilizando o modelo DOM (*Document Object Model*) (KEITH, 2005), que consiste em um modelo definido pelo *World Wide Web Consortium* (W3C) para a manipulação de documentos em formatos estruturados (*e.g.*, HTML, XML). Essa manipulação é realizada, assim como as requisições AJAX, utilizando JavaScript em conjunto com bibliotecas da linguagem que implementam o modelo DOM.

Apesar da definição do AJAX incluir a utilização da linguagem JavaScript, do formato de dados XML e do padrão DOM, há implementações do modelo que substituem essas tecnologias por alternativas equivalentes. Um caso comum desse tipo de substituição é a utilização de formatos alternativos ao XML, abordado na subseção seguinte.

2.2.1.1 AJAX sobre JSON

O JSON (*JavaScript Object Notation*) é um formato de intercâmbio de dados bastante utilizado em aplicações AJAX atualmente (MERELO-GUERVOS et al., 2008). Ele consiste em um formato de dados independente de plataforma baseado em um subconjunto da linguagem de programação JavaScript, formado por pares ordenados de nome-valor e

listas ordenadas (JSON, 1999).

As figuras 2.3 e 2.4 contém o mesmo conjunto de dados codificados, respectivamente, em XML e JSON. Comparando os dois formatos, é possível observar que a utilização do formato XML insere um maior sobrecarga de caracteres na formatação dos dados. Numa aplicação baseada em AJAX, é importante minimizar essa sobrecarga, visto que ela terá impacto em toda a comunicação realizada entre cliente e servidor. Minimizar esse impacto do formato de intercâmbio de dados na comunicação é uma das vantagens que o uso de JSON traz para o AJAX. Além disso, a utilização de AJAX em conjunto com JSON permite acesso direto aos dados representados, dispensando a complexidade normalmente associada ao acesso a dados em XML. Isso é possível pois o formato JSON é um subconjunto da linguagem JavaScript, sendo nativamente compreendido pelos interpretadores dessa linguagem. No acesso a dados codificados em XML, por outro lado, é necessário extrair os dados do documento XML que os contém, através do emprego da interface definida no modelo DOM para manipulação de dados em documentos estruturados. A utilização dessa interface requer bibliotecas específicas para esse fim, visto que tal interface, assim como o modelo DOM, não fazem parte da definição da linguagem JavaScript (RUBIO, 2007).

```
<ManagedDevice>
  <adminName>Rafael Santos Bezerra</adminName>
  <deviceName>SampleManagedRouter</deviceName>
  <hostname>123.123.123.123</hostname>
  <longitude>11.0</longitude>
  <latitude>10.0</latitude>
  <corporation>UFRGS</corporation>
</ManagedDevice>
```

Figura 2.3: Dados em XML

```
"ManagedDevice": {
    "adminName": "Rafael Santos Bezerra",
    "deviceName": "SampleManagedRouter",
    "hostname": "123.123.123.123",
    "longitude": 11,
    "latitude": 10,
    "corporation": "UFRGS"
}
```

Figura 2.4: Dados em JSON

Apesar da sua menor verbosidade e maior facilidade de acesso, o JSON não fornece diversas propriedades e vantagens do XML. Por exemplo, no tocante a interoperabilidade, XML é um formato extremamente difundido e, portanto, conta com uma base maior tanto de usuários quanto de ferramentas que o suportam, além de uma maior maturidade de padronização. Questões como essas evidenciam um *trade-off* entre XML e JSON como formatos de intercâmbio de dados. Entretanto, o JSON se apresenta como um formato de intercâmbio de dados mais apropriado para o AJAX, visto que a menor sobrecarga e a

maior simplicidade desse formato podem afetar diretamente o desempenho e o desenvolvimento de aplicações baseadas em AJAX, enquanto a interoperabilidade e o ferramental trazidos pelo XML são vantagens circunstanciais, possivelmente não aproveitadas pela maioria das aplicações baseadas em AJAX (WANG, 2011).

2.2.2 Web Services

Web Services (WS) são uma tecnologia da Computação Orientada a Serviços (*Service Oriented Computing* - SOC) (TSAI, 2005). Eles consistem em componentes de software independentes disponibilizados na Web e invocados através de protocolos baseados no modelo requisição-resposta (CURBERA et al., 2002). Esses componentes podem ser utilizados individualmente ou compostos, formando novos programas e serviços (VIANNA et al., 2007). Idealmente, eles podem ser descobertos através de um registro comum, o UDDI (*Universal Description Discovery Integration*), que, na prática, é também um Web Service. A interação desses componentes é esquematizada na Figura 2.5.

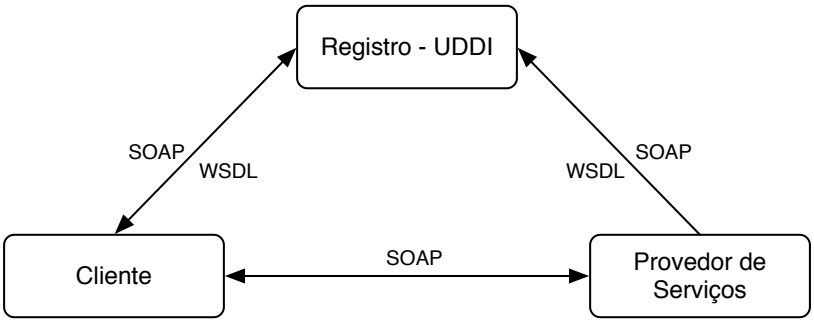

Figura 2.5: Interação dos elementos e protocolos da arquitetura Web Services.

O protocolo de invocação de serviços é chamado SOAP (*Simple Object Access Protocol*). Ele é baseado em troca mensagens XML encapsuladas em protocolos da camada de aplicação (HTTP/HTTPS/SMTP). Além disso, os Web Services contam com uma linguagem de definição, também baseada em XML, a WSDL (*Web Service Definition Language*). A utilização dessa linguagem, em conjunto com um serviço de registro como o UDDI, permite que uma aplicação encontre e acesse serviços dinamicamente, sem conhecimento *a priori* dos mesmos (CURBERA et al., 2002).

Algumas das principais características dos Web Services são:

- Modularidade: cada Web Service é um componente de software encapsulado e auto-contido;
- Independência de Plataforma: Web Services são serviços baseados em formatos independentes de plataforma, como o XML. Qualquer sistema capaz de lidar com o formato de um Web Service e capaz de se comunicar usando protocolos da camada de aplicação pode, portanto, utilizar tal serviço;
- Capacidade de Composição: Devido à sua característica modular, Web Services podem ser compostos em serviços mais complexos do que seus componentes.

Os *RESTful Web Services* têm sido pesquisados atualmente, como uma implementação alternativa de Web Services que não utiliza as tecnologias SOAP, WSDL e UDDI. A proposta de RESTful Web Services veio para se contrapor à complexidade da pilha

de protocolos padrão dos Web Services (SOAP, WSDL e UDDI), propondo Web Services de uso e implementação mais simples e diretos (PAUTASSO; ZIMMERMANN; LEYMANN, 2008). Essa implementação é baseada no modelo de arquitetura REST (*REpresentational State Transfer*), onde interações entre cliente e servidor são baseadas em operações CRUD (*Create, Read, Update, Delete*) sobre recursos disponibilizados pelo servidor. Essas operações são realizadas através da utilização de métodos do protocolo HTTP (*i.e.,* POST, GET, PUT, DELETE) equivalentes. Por exemplo, em um RESTful Web Service que tem como recurso uma lista de pessoas, a operação POST seria usada para criar uma lista de pessoas, GET para obter a lista, PUT para atualizar a lista (*e.g.,* inserir uma pessoa na lista) e DELETE para remover a lista (RODRIGUEZ, 2009). Existem também implementações de RESTful Web Services que dispensam o uso das operações PUT e DELETE. Nesse caso, a operação GET é empregada para chamadas simples aos serviços, as quais são codificadas na própria URI. Em caso de chamadas mais complexas, envolvendo, por exemplo, inserção de uma quantidade maior de dados, a operação POST é utilizada e a chamada ao serviço é codificada no campo de dados da requisição HTTP.

2.2.3 Syndication, Feeds e APIs

Syndication é um termo originado no jornalismo que significa o ato de um veículo de informação distribuir conteúdo (*e.g.,* notícias, artigos, colunas) para outros veículos. No contexto da Web 2.0, *Web Syndication* consiste em um *site* (*i.e.,* fonte) distribuir seu conteúdo para uso em outros *sites* e aplicações Web (*i.e.,* receptores) sem intervenção humana. Com essa distribuição, a fonte obtém visibilidade ao atingir os usuários do *site* receptor, enquanto o receptor se beneficia com o conteúdo recebido. *Feeds* são o mecanismo geralmente empregado no *Web Syndication*. Eles consistem em formatos padronizados para publicação de conteúdo descrito por metadados que incluem, por exemplo, *link* para a fonte do conteúdo, data e hora da publicação. Esses *feeds*, via de regra, seguem uma arquitetura *publish-subscribe* (LIU; RAMASUBRAMANIAN; SIRER, 2005), onde receptores interessados verificam periodicamente por atualizações no conteúdo disponibilizado pelo distribuidor. Uma importante característica dos *feeds* é que eles são projetados para processamento em máquina, não para consumo pelo humano. Dessa forma, é responsabilidade do receptor do conteúdo formatar as informações que a fonte disponibiliza no *feed*. Essa característica é importante para que um *site* receptor possa adequar o conteúdo dos *feeds* à sua identidade visual, além de permitir que vários *feeds* de diversas fontes sejam reunidos e processados (*i.e.,* categorizados, formatados) por aplicações denominadas *agregadores*. Os usuários finais podem, por sua vez, centralizar o acesso a conteúdo de diversos *sites* do seu interesse nesses agregadores.

Os principais formatos de *feed* utilizados atualmente são os formatos da famíla RSS (*Really Simple Syndication*, *RDF Site Summary* ou *Rich Site Summary*) (QINGCHENG; YOUMENG, 2008). O RSS é uma família de formatos de *feed* baseados em XML originados a partir do padrão *RDF Site Summary*, criado em 1999. Esse padrão de *feed* se tornou um dos mais difundidos na Internet, tendo sofrido constantes atualizações. Dentre elas a mais relevante foi o fim da conformidade com o padrão RDF (*Resource Description Framework*) (LIU; RAMASUBRAMANIAN; SIRER, 2005). Essa mudança deu origem ao RSS 2.0 e ao significado mais comum da sigla (*i.e., Really Simple Syndication*). Devido, dentre outros fatores, ao fim da conformidade com o RDF, o RSS 2.0 não é retrocompatível com as demais versões do RSS. Essa falta de retrocompatibilidade é uma das principais críticas ao RSS, entretanto, apesar dela, a maioria dos *sites* atuais emprega o RSS (QINGCHENG; YOUMENG, 2008).

APIs (*Application Programming Interfaces*) *online*, ou APIs Web, são uma forma recente de distribuir, além de conteúdo, funcionalidade e serviços entre diferentes sistemas da Web 2.0 (MAXIMILIEN; RANABAHU; GOMADAM, 2008). Essas APIs normalmente são implementadas como bibliotecas em linguagens de *scripting* para Web, como JavaScript, ou através de serviços RESTful. Através dessas APIs, sistemas da Web 2.0 expõem funcionalidades que, por sua vez, podem ser empregadas por outras aplicações da Web. Um exemplo desse caso é a funcionalidade *Like*, do *Facebook*, que permite que um usuário compartilhe uma página Web com seus pares na rede social *Facebook* através do botão *Like*, implementado na página compartilhada através da API do Facebook[1]. Outros exemplos de APIs largamente utilizadas incluem a API do Google Maps (GOOGLE, 2005), para a utilização de mapas interativos *online* e a API do Flickr[2], para compartilhamento de fotografias e imagens. O *site* ProgrammableWeb (PROGRAMMABLEWEB, 2008), um dos principais registros de APIs *online* atualmente, reporta uma quantidade de aproximadamente 5800 APIs Web em 2012. Esse número, juntamente com a ampla utilização na prática de APIs como a do Facebook e a do Google Maps, são evidências de que as APIs Web consistem em uma das principais tecnologias utilizadas na integração entre sistemas Web atualmente.

2.3 Mashups

O termo *mashup* é originário da música, onde um *mashup* é uma nova música criada a partir da combinação de diferentes peças musicais. Na Computação, esse termo descreve uma aplicação criada a partir da integração de recursos heterogêneos independentes, usualmente disponíveis na Web (MERRIL, 2003). A proposta de *mashups* é considerada uma das principais tendências de desenvolvimento de software trazidas pela Web 2.0 (O'REILLY, 2005). Contudo, várias das idéias por trás dessa proposta existiam antes mesmo do surgimento do termo Web 2.0. Dentre tais idéias, pode-se citar a integração de diversas fontes de informação. Essa idéia é relevante desde a popularização de uso dos bancos de dados e o subseqüente surgimento da necessidade de correlacionar e unir diferentes bases de dados. Essa necessidade motivou, por exemplo, a criação de ferramentas de integração de dados denominadas *data warehouses*, aplicações que processam dados de diferentes bases para disponibilizar uma única fonte unificada onde tais dados podem ser consultados (DAYAL et al., 2009). A criação de software a partir da combinação de módulos independentes e reusáveis, outra idéia básica da proposta de *mashups*, é central ao conceito de Computação Orientada a Serviço (TSAI, 2005) e está presente em tecnologias de composição de serviços, como o BPEL (ANDREWS et al., 2003). O fato dessas idéias e das tecnologias que elas originaram já existirem antes da proposta de *mashups* leva a um dos principais questionamentos feitos sobre os *mashups*, que diz respeito ao que os torna diferentes, por exemplo, de ferramentas tradicionais de integração de dados ou de composição de serviços.

Essa questão pode ser respondida através da observação de algumas características importantes de um *mashup*. A mais importante delas consiste em colocar usuário como colaborador e/ou co-desenvolvedor. Assim, espera-se que o usuário final, usualmente auxiliado por uma ferramenta de desenvolvimento de *mashups*, faça o papel de desenvolvedor de *mashups* (LIU et al., 2007). Ou seja, em um cenário ideal, um usuário final, com pouca ou nenhuma experiência em programação, usaria uma ferramenta que lhe permiti-

[1] developers.facebook.com

[2] code.flickr.com/

ria criar um *mashup* capaz de buscar as informações mais adequadas às suas necessidades na Web, combiná-las como desejar, enviá-las para serem processadas por um *Web Service* e, por fim, publicar os resultados desse processamento na Web, em um *blog* ou *wiki*, por exemplo. Esse foco no desenvolvimento de aplicações pelo usuário final é uma das principais distinções dos *mashups* em relação a tecnologias tradicionais de composição de serviços e integração de dados e/ou serviços, como o BPEL. Essas tecnologias requerem que seus usuários dominem técnicas e conceitos de desenvolvimento de software, enquanto as propostas de *mashups* objetivam eliminar essa necessidade (BEZERRA et al., 2010).

Reusabilidade, modularidade e colaboratividade também são características importantes dos *mashups*. Sob a perspectiva da Web 2.0, reutilizar dados e funcionalidades disponíveis na Web como componentes de um novo *mashup* é uma maneira de tornar esses dados e funcionalidades verdadeiramente úteis (ANKOLEKAR et al., 2008). No tocante à modularidade, o desenvolvimento de *mashups* se utiliza de princípios da Computação Orientada a Serviços (LIU et al., 2007), onde um *mashup*, tal qual um serviço, é criado como uma aplicação independente e modular, que pode vir a utilizar outros *mashups* como blocos de construção. A cooperatividade é observada quando um usuário emprega *mashups* criados por outros desenvolvedores na composição do seu próprio *mashup*, sendo estabelecida uma *cooperação pelo reuso* entre esses usuários. Tal cooperação baseada em reuso é uma maneira de aproveitar a inteligência coletiva dos usuários para a criação de um ecossistema de *mashups* cada vez sofisticados (O'REILLY, 2005).

Figura 2.6: Gráfico do crescimento do número de *mashups* registrados no *site* ProgrammableWeb.com.

O crescimento do ecossistema de *mashups* na Internet pode ser observado figura 2.6, a qual mostra que o *site* ProgrammableWeb reporta um número crescente de mais de 5000 *mashups*. O HousingMaps[1] é um exemplo de *mashup* que ilustra o potencial desse tipo de aplicação (YU et al., 2008). Ele combina anúncios de imóveis do *site* Craigslist[2] com mapas da API Web do Google Maps para exibir mapas interativos de imóveis disponíveis para venda e aluguel. O *mashup* Tubeify[3] é outro exemplo da utilização com sucesso de *mashups*. Nele, conteúdo de mídia do *site* Youtube[4] é integrado com informações

[1] www.housingmaps.com
[2] www.craiglist.com
[3] www.tubeify
[4] www.youtube.com

pessoais e regionais da rede social de mídia Last.fm[1] e do *site* BillBoard[2]. O resultado dessa integração é uma aplicação de reprodução de mídia (*i.e.*, música e vídeos) capaz de sugerir conteúdo baseando-se tanto nas preferências musicais do usuário, extraídas do Last.fm, quanto em tendências recentes, descobertas através de informações do BillBoard. Tanto o HousingMaps quanto o Tubeify são exemplos que evidenciam o uso de *mashups* para a criação de novos conhecimentos e funcionalidades a partir de recursos disponíveis na Web.

Mashups podem, de acordo com seus objetivos, ser divididos em duas grandes categorias não excludentes entre si: *mashups* de dados (BOUILLET et al., 2009) e *mashups de apresentação*. *Mashups* de dados integram fontes distintas de dados em uma lógica definida pelo usuário para lhes atribuir ou derivar deles novos significados, também servindo para codificar esses dados em outros formatos, mais adequados ao consumo humano ou de máquina, dependendo da intenção do seu criador. Já os *mashups* de apresentação têm como foco a visualização de conteúdo, onde um usuário, por exemplo, serve-se de APIs *online* de mapas para criar páginas Web com mapas interativos contendo informações obtidas em fontes como *feeds* ou *sites* de busca. Ao contrário dos *mashups* de dados, *mashups* de visualização sempre contam com uma interface de usuário, podendo utilizar inclusive *mashups* de dados como blocos de construção. Finalmente, *mashups* também podem ser divididos de acordo com seu conteúdo (*e.g.*, *mashups* de mapa, *mashups* multimídia, *mashups* de busca).

Uma das principais vantagens da proposta de *mashups* é a possibilidade da criação de soluções para necessidades *situacionais* (MAXIMILIEN; RANABAHU; TAI, 2007). Uma necessidade situacional é um problema de natureza específica e efêmera, que deve ser resolvido em tempo limitado e/ou empregando poucos recursos. Devido a essas restrições de tempo e recursos, o uso de uma solução *ad hoc* para o problema é geralmente a abordagem mais adequada. Esse tipo de solução pode ser implementada através de *softwares* de pequeno porte denominados *aplicações situacionais*. As principais caraterísticas de uma aplicação situacional incluem (MOHAMMADI; KHALILI; ASHOORI, 2009):

- **Escopo limitado**: diferentemente de ferramentas de grande porte, cujo objetivo é resolver uma ou mais classes de problemas, aplicações situacionais resolvem um problema específico como, por exemplo, a criação de um determinado tipo de relatório. Um escopo limitado não significa que ele seja fechado, pelo contrário, o mesmo pode ser extremamente mutável, pois o conjunto de requisitos da aplicação pode sofrer influências externas. O desenvolvimento de uma aplicação situacional precisa ser capaz de suportar mudanças, o que demanda agilidade, modularidade, reusabilidade e a possibilidade de descarte da aplicação sem maiores prejuízos;

- **Conjunto restrito de usuários-alvo**: o conjunto de usuários-alvo de uma aplicação situacional é limitado aos indivíduos que estão lidando com o problema abordado, usualmente uma pessoa ou uma equipe. Entretanto, outros usuários que enfrentem o mesmo problema, ou um problema da mesma categoria, podem vir a reutilizar uma mesma aplicação situacional;

- **Ciclo de vida variável**: o ciclo de vida de uma aplicação situacional é normalmente curto, pois a necessidade para a qual a aplicação se destina é temporária. Entretanto,

[1] www.last.fm
[2] www.billboard.com

esse ciclo de vida pode se estender indefinidamente, caso a aplicação resolva um problema cotidiano e/ou recorrente. Tal característica altamente variável do ciclo de vida de aplicações situacionais enfatiza a necessidade das mesmas apresentarem baixo custo de implementação e descarte, bem como alta reusabilidade;

- **Baixo tempo de desenvolvimento inicial**: o desenvolvimento de uma aplicação situacional é marcado por um baixíssimo tempo de implementação inicial, destinado a criar uma solução apenas suficiente para o problema abordado. A partir dessa implementação, pode-se desenvolver melhorias e readaptações caso, por exemplo, seja necessário reagir ao surgimento de novos requisitos. É uma idéia análoga ao "beta teste perpétuo"da Web 2.0 e alinhada com práticas de desenvolvimento de *software* ágil.

Por sua característica dinâmica e voltada à integração ágil, os *mashups* se apresentam como uma possível forma de materializar aplicações situacionais, que tem como vantagem a aproximação dos usuários-alvo da criação da aplicação situacional (MOHAMMADI; KHALILI; ASHOORI, 2009). O *mashup* Sinsai Info[1] é um exemplo recente de *mashup* situacional, criado para abordar necessidades temporárias de vítimas de terremotos ocorridos no Japão em 2011. Ele consiste em um mapa de ajuda a vítimas, criado através de uma API *online* de mapas, o qual foi integrado com informações como localização de abrigos, centros de evacuação e atendimento médico, extraídas de diversas fontes. No contexto do gerenciamento de redes, é possível identificar algumas necessidades situacionais como, por exemplo, reação a ataques e/ou intrusões na rede, que são eventos temporários e com características únicas (DOS SANTOS et al., 2010).

Para viabilizar um cenário onde qualquer usuário possa resolver problemas situacionais através de *mashups*, é necessário que haja uma infra-estrutura que possibilite a criação de *mashups* com agilidade, facilidade e dinamismo. Isso é possível através de ferramentas cujo propósito é viabilizar a criação de *mashups* por usuários inexperientes em desenvolvimento de *software*. Essas ferramentas são denominadas **sistemas de *mashup*** e são abordadas em detalhes na próxima seção.

2.3.1 Sistemas de *Mashups*

Um sistema de *mashups* é uma ferramenta cujo objetivo é permitir a criação dinâmica e facilitada de *mashups* pelo usuário final, através da composição de recursos disponíveis na Web (LIU et al., 2007). Sistemas de *mashups* são geralmente sistemas Web que funcionam de forma análoga a IDEs (*Integrated Development Environment*) para linguagens de programação, fornecendo funcionalidades que auxiliam e agilizam o processo de desenvolvimento. Dentre essas funcionalidades pode-se citar, por exemplo, elementos visuais que facilitem a definição de um *mashup* pelo usuário e mecanismos que permitem que o usuário disponibilize seus *mashups* na Web. Os sistemas de *mashup*, entretanto, têm responsabilidades que vão além do esperado de uma IDE, visto que eles também executam as composições e armazenam os *mashups* criados. Isso leva a um cenário misto onde o sistema de *mashups* é ao mesmo tempo ambiente de desenvolvimento, ambiente de execução e repositório de *mashups*.

Além das três funções básicas (*i.e.*, criação, execução e armazenamento de composições), sistemas de *mashups* também costumam agregar outras funções como, por exemplo, o compartilhamento de *mashups*. É através dessa função que o sistema de *mashups*

[1] www.sinsai.info/ushahidi/index.php/

possibilita que um usuário utilize *mashups* criados por terceiros como base para criar composições mais sofisticadas (ENNALS; GAY, 2007). Tal compartilhamento pode, inclusive, acontecer entre usuários de sistemas de *mashups* distintos, desde que tais sistemas possuam mecanismos adequados para tal fim. Nesse cenário, um *mashup* criado em um sistema (*i.e.*, sistema de origem) pode ser executado em outro (*i.e.*, sistema receptor) (DOS SANTOS et al., 2010), desde que o sistema de origem forneça uma cópia da composição compartilhada ao receptor. Também é possível, no compartilhamento entre sistemas de *mashups* distintos, que o *mashup* compartilhado seja apenas invocado pelo sistema receptor. Nesse caso, a composição é executada no sistema de origem e seus resultados são enviados ao receptor.

Outra importante função normalmente desempenhada pelos sistemas de *mashups* é o controle de acesso às composições, que permite que um usuário e/ou um administrador do sistema definam regras de acesso aos *mashups* criados no sistema. Essas regras podem, por exemplo, representar quais *mashups* podem ser acessados por cada usuário do sistema de *mashups* e quais composições serão compartilhadas com sistemas de *mashup* externos. Finalmente, sistemas de *mashups* também podem disponibilizar funcionalidades sociais, como a categorização de *mashups* por *tags* aplicadas pelos usuários e *ranking* de *mashups* mais utilizados (ANKOLEKAR et al., 2008). Esse tipo de funcionalidade visa enriquecer o ecossistema de *mashups* com informações de natureza semântica (*e.g.*, dados relativos a qualidade materializados no *ranking* e dados indicando características dos *mashups* em formato de *tags*) fornecida pelos próprios usuários finais.

Para cumprir seu objetivo, sistemas de *mashups* têm como requisito uma alta usabilidade (HOYER et al., 2008). Suas interfaces de composição devem apresentar abstrações de alto nível para o processo de composição de recursos heterogêneos, permitindo que esse processo seja desempenhado por um usuário leigo em desenvolvimento de software. A viabilidade dessas abstrações depende da automatização de várias questões técnicas pelo sistema de *mashups*. Dentre tais questões, pode-se citar o acesso a recursos heterogêneos através de diversos protocolos (*e.g.*, SOAP, HTTP) e a manipulação de dados em diferentes formatos (*e.g.*, RSS, CSV) (BANERJEE; DASGUPTA; MUKHERJEA, 2007). Para viabilizar esse cenário, a maioria dos sistemas de *mashup* segue a arquitetura básica representada na figura 2.7.

A primeira etapa exibida na figura 2.7 é denominada etapa de interação. Ela abrange a interação do usuário com o sistema de *mashups*. Essa interação é realizada através de duas interfaces com o usuário distintas, a interface de composição e a interface de execução. Visto que sistemas de *mashup* são normalmente aplicações Web, tais interfaces normalmente são implementadas no lado cliente da aplicação, executado no navegador Web do usuário. Isso acontece pois a maioria das tecnologias criação de interfaces Web ricas (*e.g.*, AJAX, Flash) emprega *client-side scripting* para ampliar as capacidades gráficas de uma aplicação Web (THOMAS, 2008).

Na etapa de interação, os usuários são divididos em dois perfis conceituais, desenvolvedores e consumidores (YU et al., 2008). Cada um desses perfis interage com uma das interfaces do sistema. Entretanto, apesar dessa divisão, o mesmo humano pode desempenhar ambos os papéis, interagindo com diferentes interfaces em momentos distintos. Nesse cenário, o desenvolvedor é o usuário interessado em compor recursos para criar *mashups* utilizando a interface de composição. Essa interface possibilita a criação de *mashups* através da manipulação de elementos visuais ou textuais que abstraem detalhes técnicos na composição de recursos heterogêneos. Interfaces de composição geralmente são baseadas em modelos de interação com o usuário amigáveis, como o *drag'n drop*.

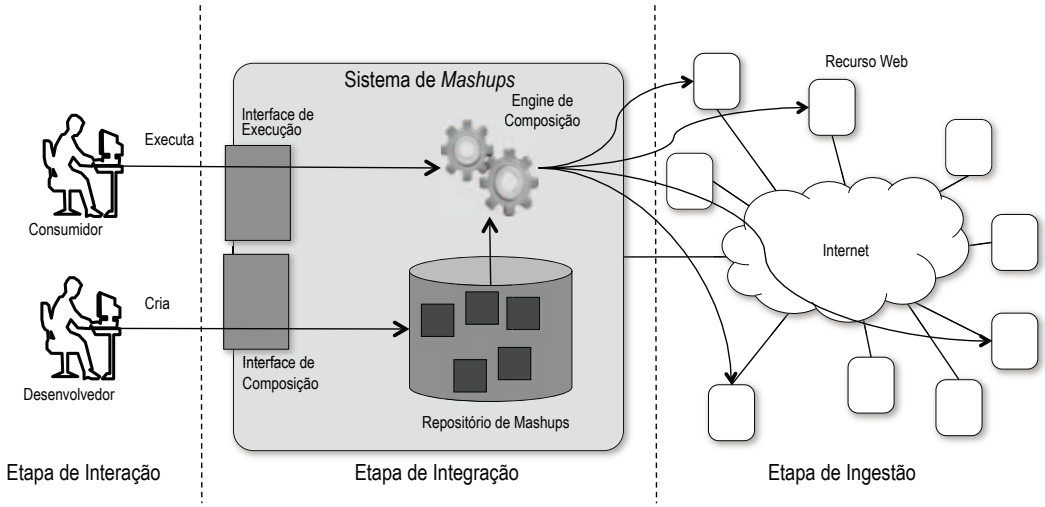

Figura 2.7: Arquitetura básica de um sistema de *mashups*.

O consumidor, por sua vez, é o usuário interessado em acessar os resultados das composições de recursos. Para isso, esse consumidor interage com a interface de execução, a qual permite que esse usuário busque, recupere e utilize *mashups* que lhe interessem. A maioria dos sistemas de *mashups* também permite que *mashup* sejam publicados como páginas Web, podendo ser acessados diretamente através de um endereço único (*i.e.*, uma URL). Isso torna a interface de execução transparente para o consumidor, desde que ele tenha conhecimento prévio do endereço do *mashup*.

Na segunda etapa da arquitetura, denominada etapa de integração, o sistema de *mashups* realiza três tarefas distintas: validação, armazenamento e execução de *mashups*. As ações executadas na tarefa de validação dependem da implementação do sistema de *mashups*, podendo variar desde a verificação de compatibilidade entre entradas e saídas dos componentes até verificações mais elaboradas, como, por exemplo, disponibilidade dos componentes empregados e permissões de acesso a determinados componentes/*mashups* (SANTOS et al., 2011). Após a validação, é realizada a tarefa de armazenamento, na qual a composição criada é armazenada no repositório de *mashups*. Os dados armazenados nessa tarefa consistem na descrição do *mashup* criado em um formato conhecido pela *engine* de composição, elemento da arquitetura responsável pela tarefa de execução de *mashups*.

As tarefas de validação e armazenamento da etapa de integração são realizadas de forma consecutiva e, ao final delas, têm-se um *mashup* armazenado no repositório de *mashups* do sistema. Esse cenário viabiliza a terceira tarefa da etapa de integração, a execução dos *mashups*. Nela, a *engine* de composição recupera *mashups* armazenados no repositório e materializa a composição de recursos, integrando componentes de acordo com a lógica definida pelo desenvolvedor do *mashup* executado. Tais componentes podem ser tanto componentes internos do sistema (*e.g.*, operadores de dados, operadores lógicos) quanto recursos externos. Esses últimos são acessados na etapa de ingestão.

Na terceira e última etapa da arquitetura, denominada etapa de ingestão (HOYER et al., 2008), é realizada a adaptação dos recursos externos que vão servir de componentes para os *mashups* (YU et al., 2008). Tal adaptação é desempenhada pelos *wrappers*, que são adaptadores para recursos disponíveis na Web. Cada *wrapper* é responsável por se

comunicar com o recurso a ser adaptado, lidando com os protocolos, formatos de dados e métodos de acesso necessários. Além da comunicação, os *wrappers* são responsáveis por permitir que esses recursos sejam acessados pelo sistema de *mashups* através de uma interface única e disponibilizando dados provenientes dos recursos externos em um formato canônico conhecido pelo sistema. Espera-se que o modelo de *wrappers* de um sistema de *mashups* permita que desenvolvedores de *software* criem novos adaptadores e os integrem no sistema dinamicamente, de forma similar aos mecanismos de *plugin* utilizados em outros contextos para permitir a adição de novas funcionalidades em um *software*.

Exemplos de adaptadores/*wrappers* incluem:

- Adaptadores de protocolo: permitem que o sistema de *mashups* se comunique através de diferentes protocolos. Adaptadores SNMP e Telnet são dois casos úteis no gerenciamento de redes;

- Adaptadores para APIs *online*: padronizam o acesso a recursos da Web disponibilizados através de APIs, como serviços de mapa (*i.e.*, Google Maps), *microblogging* (*i.e.*, Twitter), redes sociais (*i.e.*, Facebook), serviços de geolocalização, dentre outros;

- Adaptadores de tipos de dados: permitem que o sistema de *mashups* manipule dados em diferentes formatos, como XML, CSV (*Comma Separated Values*) e XLS (*Excel Spreadsheet*);

- Adaptadores de ferramentas: possibilitam a comunicação do sistema com outros *softwares*. Nesta categoria se incluem, por exemplo, adaptadores a SGBDs (Sistema de Gerenciamento de Bancos de Dados), adaptadores de acesso a *softwares* de relacionamento com o cliente (*Consumer Relationship Management* - CRM) e ferramentas de *tickets* de solicitações de usuários.

Como é possível observar pela descrição da arquitetura, existem diversos desafios na criação de sistemas de *mashups* (YU et al., 2008). Do ponto de vista da interação com o usuário, pode-se citar como exemplo a definição de um modelo de criação de *mashups* adequado a usuários sem experiência em desenvolvimento de *software* e, ao mesmo tempo, suficientemente expressivo para permitir a composição de diversos recursos disponíveis na Web. A característica heterogênea desses recursos também traz desafios como, por exemplo, a padronização de métodos de acesso e formatos de dados diversos, além da integração de dados heterogêneos. Outro desafio trazido por esses recursos é a verificação de disponibilidade, visto que os mesmos estão na Web e, geralmente, oferecem poucas garantias de funcionamento. Existem, ainda, os desafios de escalabilidade de um sistema de composição de recursos e execução de *composições* utilizado por um grande número de usuários. Finalmente a manutenção do ecossistema de *mashups* oferece desafios como o armazenamento desses *mashups* e o controle de acesso aos mesmos. Ainda assim, existem várias iniciativas no sentido de criar sistemas de *mashups*. Algumas das principais serão discutidas na próxima subseção.

2.3.2 Sistemas de *Mashups* Atuais

O Yahoo! Pipes[1] é um dos principais sistemas de *mashups* disponível na Web (YU et al., 2008) atualmente. Através da interface de composição de *mashups* desse sistema,

[1] pipes.yahoo.com

é possível acessar dados em diferentes formatos de dados (*e.g.*, RSS, XML, JSON), realizar operações sobre esses dados e publicar *mashups* em formatos como *feeds* RSS, mapas e páginas de busca personalizadas. Para isso, o usuário conecta visualmente módulos que representam recursos, operações ou outros *mashups* através de conexões lógicas denominadas *pipes*, por onde acontece o fluxo dos dados. O Yahoo! Pipes mantém um ecossistema de dezenas de milhares de *mashups* criados dessa forma. Além do Yahoo!, o Google e a Microsoft também conduziram iniciativas envolvendo sistemas de *mashups*, respectivamente o Google Mashup Editor[1] e o Microsoft Popfly (YU et al., 2008). Entretanto, ambas foram descontinuadas. Finalmente, existem também sistemas de *mashups open-source* disponíveis na Web. Os principais deles são abordados a seguir:

- **Apatar**[2]: É um sistema de *mashups* materializado em uma aplicação *desktop*, que é instalada e executada na máquina do usuário. Seu objetivo é permitir a criação *mashups* de dados para uso em ambientes pessoais e empresariais através de uma interface de conexão de módulos visuais similar à do Yahoo! Pipes. O Apatar também disponibiliza módulos de integração com outras ferramentas, incluindo editores de planilha (*e.g.*, Microsoft Excel) e sistemas de CRM;

- **WSO2 Mashup Server**[3]: É um sistema de *mashups* baseado em *scripting* que emprega a linguagem JavaScript e permite publicar *mashups* como Web Services SOAP e RESTful. O WSO2 disponibiliza bibliotecas que facilitam o acesso a Web Services através de chamadas criadas em JavaScript, automatizando questões técnicas envolvidas no acesso a esses serviços (*e.g.*, criação de mensagens de solicitação SOAP, extração de dados das respostas dos serviços). Além disso, o sistema automatiza a publicação do *mashup* do usuário como um Web Service realizando, por exemplo, a criação automática de uma WSDL para o Web Service criado.

Na indústria, a aplicação da tecnologia de *mashups* se dá através de sistemas de *enterprise mashups*. Esses sistemas visam possibilitar que colaboradores em uma empresa resolvam problemas operacionais através da criação de aplicações extremamente específicas para esses problemas, denominadas *enterprise mashups* (HOYER et al., 2008), as quais podem integrar tanto sistemas disponíveis na rede interna da empresa quanto recursos disponíveis na Web. Um sistema de *enterprise mashups* de uma empresa pode ser mantido internamente, pelo departamento de Tecnologia da Informação (TI), ou pode ser administrado uma outra empresa prestadora de serviços. Um exemplo de sistema de *enterprise mashups* é o Kapow Katalyst[4], que consiste em uma ferramenta de integração de recursos cujo foco é extração automatizada de dados de diversas fontes não estruturadas e criação, em tempo real e através de linguagem visual, de *workflows* que automatizam o processamento e a integração desses dados. O JackBe Presto[5] tem um foco similar, permitindo, adicionalmente, a publicação de *mashups* como *aplicativos* executáveis tanto em *desktops* quanto em dispositivos móveis.

Entre os sistemas de *enterprise mashups*, é importante também mencionar o IBM *Mashup Center*[6] como um dos principais esforços. Sua proposta é, além de integrar recursos externos disponíveis na Web, suportar a integração com diversas ferramentas de

[1] code.google.com/gme/
[2] www.apatar.com/
[3] wso2.com/products/mashup-server/
[4] kapowsoftware.com/products/kapow-katalyst-platform/index.php
[5] www.jackbe.com/
[6] www-01.ibm.com/software/info/mashup-center/

software empresarial do catálogo da IBM. Para tal, o IBM *Mashup Center* permite a integração visual de *feeds* de dados e *widgets*, que são *softwares* de pequeno porte independentes entre si executados no navegador Web do cliente. Nesse caso, os *widgets* podem ser empregados, por exemplo, para controlar interações entre um *mashup* e seu usuário (*i.e.*, onde o *widget* contém uma interface visual para tal) ou para encapsular recursos externos como APIs *online* (*i.e.*, onde o *widget* tem função similar a um *wrapper*). A ferramenta também suporta tanto *feeds* de dados estruturados, como arquivos XML ou dados obtidos via SQL (*Structured Query Language*), quanto dados não estruturados obtidos em páginas Web. Além disso, o sistema também tem funcionalidades voltadas ao gerenciamento de ecossistemas de *mashups*, *widgets* e *feeds*, como, por exemplo, catalogação desses recursos.

Vista a fundamentação teórica sobre *mashups*, bem como os problemas que sua aplicação pretende resolver, a próxima subseção tratará da aplicação de *mashups* no gerenciamento de redes. Serão discutidos também possíveis recursos que um administrador de redes pode integrar nos seus *mashups* atualmente, sejam eles ferramentas de gerenciamento com interfaces adequadas ao uso em *mashups* ou ferramentas de propósitos gerais já disponíveis na Web.

2.4 Mashups no Gerenciamento de Redes

Existem potenciais benefícios em aplicar tecnologias e princípios da Web 2.0 no gerenciamento de redes através dos *mashups*. O principal desses benefícios é permitir que um administrador leigo em desenvolvimento de software de redes crie novas aplicações através da composição de diferentes ferramentas de gerenciamento. Ao possibilitar esse tipo de composição, a aplicação de *mashups* se mostra como uma possível solução para o problema das múltiplas ferramentas discutindo anteriormente.

Além do problema das múltiplas ferramentas, administradores de rede também enfrentam necessidades situacionais (MAXIMILIEN; RANABAHU; TAI, 2007) que emergem no cotidiano do gerenciamento de uma rede. Por exemplo, pode-se citar o diagnóstico de falhas na rede ou a verificação de que um determinado acordo de nível de serviço (*Service Level Agreement* - SLA) está sendo cumprido. Os administradores normalmente lidam com essas situações de forma *ad hoc* ou empregando *scripting*. Os *mashups*, no entanto, apresentam determinadas características que os tornam uma alternativa mais apropriada para lidar com essas necessidades situacionais, tais como agilidade na integração de recursos e baixo custo de alteração ou descarte de composições já existentes (MOHAMMADI; KHALILI; ASHOORI, 2009).

O fato de que diversas ferramentas de gerenciamento atuais expõem dados e funcionalidades através de interfaces Web é um forte indício de que o gerenciamento de redes pode explorar os benefícios trazidos pelos *mashups* (DOS SANTOS et al., 2010). A seguir, serão abordados alguns dos exemplos dessas ferramentas e como elas podem ser utilizadas em *mashups*:

- **Multi-Router Traffic Grapher (MRTG)** - o MRTG é uma ferramenta Web largamente usada no gerenciamento de redes. Ele é capaz de criar gráficos baseados em informações extraídas em tempo real de elementos gerenciados da rede. Na sua configuração mais comum, o MRTG mostra histogramas de tráfego criados a partir de informações de roteadores obtidas em tempo real via consultas SNMP. Os gráficos criados pelo MRTG são geralmente exibidos em páginas Web, podendo ser extraídos utilizando *screen scraping* (BEZERRA et al., 2010). Após essa extração,

os gráficos do MRTG podem ser facilmente integrados em *mashups* de visualização de informações de redes gerenciadas. Além disso, os arquivos de *log* gerados pelo MRTG podem ser processados por *parsers* de forma a obter informações sobre o tráfego da rede. Alguns exemplos de organizações que utilizam essa ferramenta incluem a Rede Nacional de Ensino e Pesquisa[1] (RNP) e a rede acadêmica de pesquisa da Suiça, denominada SWITCH[2];

- **Cisco NetFlow**[3] - o Netflow é uma ferramenta usada para capturar informações relativas a tráfego IP e agregá-las em fluxos de rede (*Network Flows - NetFlows*). Apesar do NetFlow ser uma tecnologia originalmente promovida pela Cisco, várias ferramentas gratuitas podem ser usadas para analisar e visualizar fluxos de rede. As ferramentas *FlowScan e NTop*, por exemplo, permitem a criação de grafos e visualizações a partir de fluxos previamente analisados. Tal como o MRTG, os gráficos gerados por essas ferramentas podem ser exibidos como páginas Web e, portanto, integrados em *mashups* através do uso de *screen scraping* (MERRIL, 2003). A Cisco e a Universidade de Houston[4] são exemplos de instituições que empregam o NetFlow no gerenciamento das suas redes;

- **APIs Web de gerenciamento de equipamentos** - atualmente, vários fabricantes de equipamentos de rede definem APIs proprietárias através das quais várias funções de gerenciamento (*e.g.*, diagnóstico, configuração) podem ser utilizadas. Por exemplo, tanto a Cisco[5] quanto a Juniper[6] definem APIs de gerenciamento baseadas em documentos XML trafegando sobre protocolos Web como HTTP e SOAP. O funcionamento dessas APIs é similar ao de APIs *online* largamente utilizadas em *mashups* atuais (*e.g.*, a API Youtube[7]) (YU et al., 2008). APIs Web de gerenciamento como as mencionadas podem, portanto, ser usadas diretamente em *mashups*;

- **Looking Glass** - Instituições que administram redes de grande porte (*e.g.*, universidades, provedores de serviço e empresas de telecomunicação) normalmente disponibilizam *sites* Web denominados *Looking Glasses*, com o objetivo de informar usuários avançados e instituições interessadas sobre o estado da rede administrada. Esses *sites* geralmente contém informações relativas ao *backbone* da rede e oferecem várias ferramentas de gerenciamento como, por exemplo, *ping* e *traceroute*. Como essas informações e ferramentas são disponibilizadas via interfaces Web, elas são recursos com os quais é possível compor um *mashup*. A RNP é um exemplo de instituição que disponibilza um *Looking Glass*[8] da sua rede.

Além das ferramentas de gerenciamento listadas, o uso de *mashups* também permite que administradores criem composições mais sofisticadas empregando, para tal, recursos Web que não foram concebidos para o gerenciamento de redes. A seguir, são listados alguns desses recursos e como eles podem ser úteis no gerenciamento:

[1] http://www.rnp.br/
[2] http://www.switch.ch/
[3] http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html
[4] http://www.uh.edu/
[5] http://www.cisco.com/en/US/docs/net_mgmt/ip_solution_center/4.0/developer/guide/apipg.html
[6] http://www.juniper.net/support/products/xmlapi/
[7] https://developers.google.com/youtube/2.0/developers_guide_protocol
[8] http://www.rnp.br/servicos/lg/

- **Serviços de Geolocalização de IP** - esses serviços Web são capazes de, dado um determinado endereço IP, fornecer informações geográficas sobre a localização do mesmo (*e.g.*, latitude, longitude, país, cidade) (GIACCARDI; FOGLI, 2008). O IP2Location[1] é um exemplo de serviço de geolocalização. Esse tipo de serviço é tradicionalmente utilizado por administradores de *sites* Web para localizar os visitantes das suas páginas. No gerenciamento de redes, tais serviços podem ser usados em conjunto com ferramentas de gerenciamento (*e.g.*, NetFlow) para gerar estatísticas geográficas de tráfego (KAMOUN, 2005), permitindo que um administrador de redes identifique em tempo real quais são os países de origem e destino do tráfego da rede. Além disso, a geolocalização pode ser usada para propósitos de segurança como, por exemplo, identificar o país de origem de um ataque à rede;

- **APIs *online* de Mapas** - essas APIs possibilitam a criação mapas interativos e a publicação desses mapas na Web. Exemplos desses serviços de mapas incluem o Google Maps e o OpenStreetMap[2]. Ao empregar essas APIs em *mashups*, administradores podem criar mapas de rede interativos e geolocalizados. Além disso, mapas de rede criados com APIs online são geralmente aplicações Web, as quais são nativamente independentes de plataforma, facilmente publicáveis e compartilháveis (DOS SANTOS et al., 2010).

Como abordado nessa seção, possibilitar que administradores de redes criem *mashups* pode ser uma solução tanto para problemas envolvendo múltiplas ferramentas de gerenciamento quanto para necessidades situacionais. Além disso, é possível observar que já há, atualmente, tanto ferramentas de gerenciamento quanto recursos da Web que podem ser potencialmente explorados por *mashups* no contexto do gerenciamento. Ou seja, atualmente existem tanto situações de gerenciamento onde o uso de *mashups* pode ser benéfico quanto os recursos necessários para a aplicação dessa tecnologia. Entretanto, até a redação do presente trabalho, não há outros esforços abordando a aplicação de *mashups* no gerenciamento de redes.

Uma das principais dificuldades na aplicação de *mashups* no gerenciamento de redes consiste no fato de que a criação de *mashups* de forma *ad hoc* (*i.e.*, sem o auxílio de uma ferramenta de gerenciamento) é um processo de desenvolvimento de *software* tão ou mais complexo do que a criação de *scripts* (BOUILLET et al., 2009). Essa complexidade é incompatível com o objetivo de permitir que administradores de rede criem *mashups* já que os mesmos são, geralmente, leigos em desenvolvimento de *software*. Uma possível forma de eliminar essa complexidade consiste no emprego de um *sistema de mashups* voltado para o gerenciamento de redes. O capítulo seguinte discutirá uma arquitetura de referência para tal sistema. Essa arquitetura contempla todas as etapas do funcionamento de um sistema de *mashups* descritas na seção anterior (*i.e.*, ingestão, integração e apresentação), propondo também uma classificação dos componentes utilizados na criação de *mashups* de gerenciamento.

[1] http://www.ip2location.com/

[2] http://www.openstreetmap.org/

3 ARQUITETURA DE UM SISTEMA DE *MASHUPS* DE GERENCIAMENTO DE REDES

O presente trabalho investiga a aplicação de *mashups* no gerenciamento de redes através do emprego de um sistema de *mashups* voltado a administradores de redes. Tal sistema é o elemento que aproxima esse administrador da criação de novas ferramentas. Para tal, o sistema deve possibilitar que um administrador de redes, conceitualmente leigo em desenvolvimento de *software*, crie ***Mashups* de Gerenciamento**, os quais consistem em novas ferramentas de gerenciamento de redes criadas a partir da composição de recursos disponíveis tanto na rede gerenciada (*e.g.*, ferramentas de gerenciamento já existentes) quanto disponíveis na Internet. Ou seja, empregando o sistema de *mashups*, esse administrador deve ser capaz, por exemplo, de criar uma composição que colha informações de equipamentos da rede via SNMP, correlacione essas informações com descrições estruturadas de SLAs e crie uma representaçào visual do resultado da correlação. O presente capítulo descreve uma arquitetura para tal sistema.

A principal responsabilidade do sistema de *mashups* de gerenciamento de redes é permitir que o administrador concentre-se apenas na definição da lógica das suas composições. Para isso, esse sistema é responsável por lidar com questões técnicas envolvidas nessas composições (*e.g.*, comunicação com diferentes protocolos, *parsing* de diferentes tipos de dados) de forma automatizada e transparente para o usuário final (MERRIL, 2003). Outra responsabilidade importante do sistema de *mashups* de gerenciamento é possibilitar o compartilhamento de *mashups* entre administradores de rede. Tal compartilhamento deve possibilitar que um administrador utilize uma composição criada por outro tanto como uma ferramenta *standalone* quanto como componente em um novo *mashup* de gerenciamento mais sofisticado.

A arquitetura para sistemas de *mashups* de gerenciamento de redes está representada na figura 3.1. A decisão de criar essa arquitetura foi tomada após ser verificado que os sistemas de *mashups* disponíveis atualmente não atendiam a requisitos trazidos tanto pelo gerenciamento de redes (*e.g.*, adaptabilidade a ferramentas UNIX executadas na máquina do usuário) quanto pela natureza da pesquisa realizada (*e.g.*, facilidade em implementar de mudanças e propor novos tipos de componentes). A arquitetura, no entanto, usa como base os vários sistemas de *mashups* analisados durante o andamento da pesquisa, em especial sistemas *open source* como o Apatar e o WSO2, abordados no capítulo anterior. Conceitos propostos por esforços acadêmicos referentes a *mashups* (YU et al., 2008) (BANERJEE; DASGUPTA; MUKHERJEA, 2007) também foram utilizados na definição da arquitetura. Devido a essa base conceitual de trabalhos acadêmicos e tecnologias não originalmente concebidos visando o gerenciamento de redes, a arquitetura proposta pode ser extrapolada para sistemas de *mashups* de escopo geral, ainda que o objetivo da mesma

seja atender necessidades de administradores de redes.

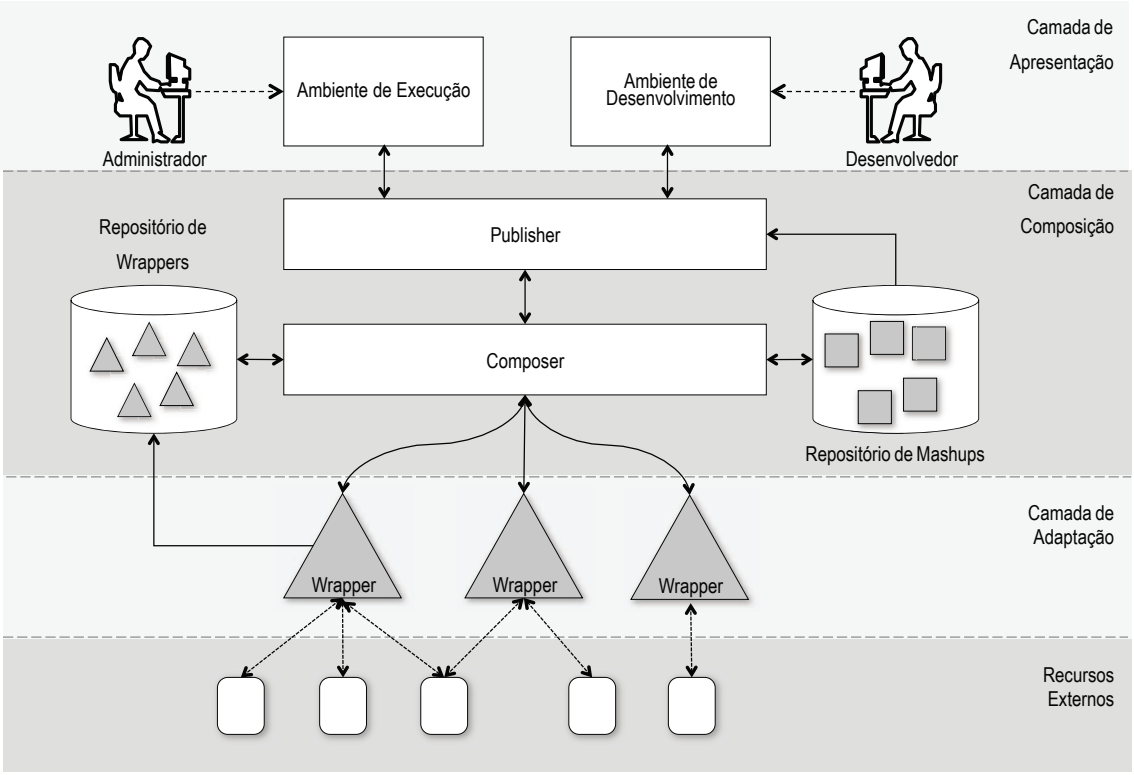

Figura 3.1: Arquitetura proposta para um sistema de *mashups* de gerenciamento de redes

Como é possível observar pela figura 3.1, a arquitetura é baseada no modelo de três camadas (*i.e.*, dados, lógica e apresentação) (ECKERSON, 1995). Esse é um modelo para sistemas Web validado pela academia, o qual proporciona o desacoplamento dos elementos lógicos do sistema modelado. Por exemplo, no caso específico de um sistema de *mashups*, esse desacoplamento permite que mudanças nos recursos externos sejam tratadas apenas na camada de adaptação. Na arquitetura, essa camada de adaptação corresponde à camada de dados do modelo proposto por (ECKERSON, 1995). Ela é responsável por adaptar os recursos externos dos quais os dados usados nos *mashups* serão extraídos. A camada lógica do modelo, por sua vez, é denominada camada de composição na arquitetura, sendo responsável por tratar da lógica envolvida na composição dos *mashups*. Essa camada também cuida das regras de negócio envolvidas no gerenciamento e compartilhamento das composições criadas. Finalmente, a camada de apresentação da arquitetura controla a interação dos usuários com o sistema de *mashups*, tal como no modelo de três camadas.

Os usuários do sistema de *mashups* de gerenciamento de redes são divididos em dois perfis: **administrador** de redes e **desenvolvedor** de mashups. O administrador é o usuário interessado em utilizar *mashups* de gerenciamento existentes para resolver necessidades da gerência de uma rede. O desenvolvedor, por sua vez, é o usuário interessado em criar novos *mashups* para resolver suas necessidades de gerenciamento. Essa divisão de papéis é puramente conceitual e seu objetivo é permitir uma melhor compreensão dos requisitos de diferentes administradores de redes ao usar o sistema de *mashups* de gerenciamento. Entretanto, num cenário ideal da aplicação desse sistema, é importante que o mesmo

administrador de rede desempenhe ambos os papéis de acordo com suas necessidades e com os *mashups* disponíveis.

Existe ainda um terceiro perfil envolvido na operação de um sistema de *mashups* de gerenciamento: o de criador de *wrappers*. Ele é o humano responsável por desenvolver adaptadores para os recursos externos a serem utilizados nas composições. Esse perfil demanda habilidade em desenvolvimento de *software* (LIU et al., 2007) e não requer conhecimentos em gerenciamento de redes. Uma discussão detalhada sobre o criador de *wrappers* foge ao escopo dessa dissertação, visto que esse perfil é adequado para especialistas em desenvolvimento de *software*, não para administradores de rede. Para os propósitos desse trabalho, a arquitetura assume que o sistema de *mashups* possui os *wrappers* apropriados para os recursos utilizados nas composições.

Nessa seção, foram discutidos os conceitos gerais referentes à arquitetura. As próximas seções do capítulo abordarão essa arquitetura em detalhes. Para tal fim, cada camada terá seus componentes abordados em uma seção distinta, seguindo uma lógica *bottom-up*, que permite uma compreensão mais aprofundada dos conceitos apresentados. Após a discussão de cada camada, a seção seguinte abordará a interação entre essas camadas e seus componentes durante a operação do sistema de *mashups* de gerenciamento. Finalmente, a última seção do capítulo tratará sobre os elementos de interação que o desenvolvedor pode utilizar para compor seus *mashups*.

3.1 Camada de Adaptação

O objetivo da camada de adaptação é adaptar os recursos externos que serão utilizados nos *mashups* de gerenciamento, permitindo que o sistema de *mashups* acesse tais recursos de forma padronizada (YU et al., 2008). Essa padronização possibilita que o sistema lide com os diversos protocolos (*e.g.*,HTTP, SNMP), formatos de dados (*e.g.*, CSV, XML), recursos Web (*e.g.*, APIs *online*) e ferramentas de gerenciamento (*e.g.*, MRTG, Netflow) utilizados por administradores de redes.

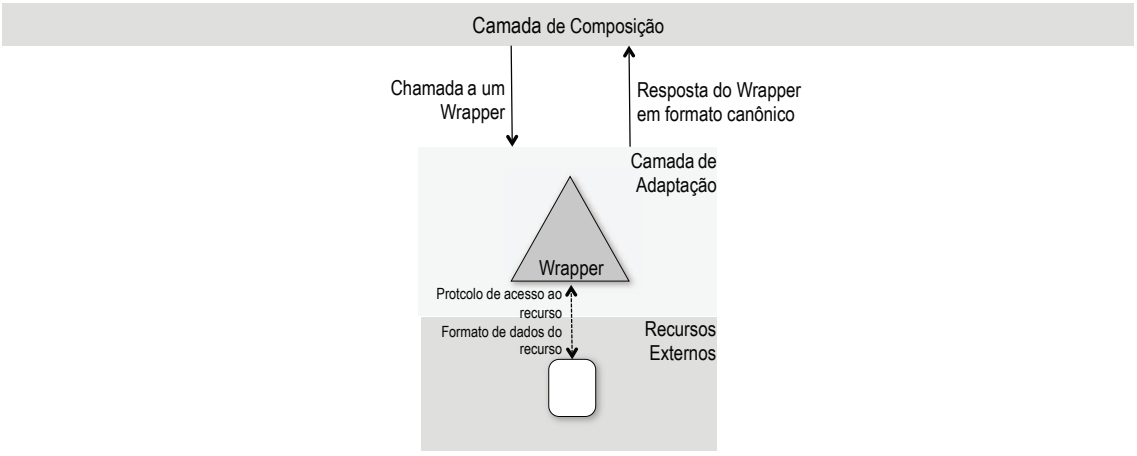

Figura 3.2: Funcionamento da camada de adaptação

O objetivo definido acima é alcançado através do emprego de adaptadores para os recursos externos denominados *wrappers* (NAKANO et al., 2007). Um *wrapper* é um elemento de *software* independente, capaz de comunicar-se com um determinado recurso

(*e.g.,* uma ferramenta de gerenciamento ou uma página Web) e adaptar o acesso a tal recurso para um método padronizado conhecido pelo sistema de *mashups*. Essa adaptação está esquematizada na figura 3.2. Nela, é possível observar que quando o sistema precisa interagir com recursos externos, a camada de composição do sistema de *mashups* realiza chamadas aos *wrappers*. Esses *wrappers*, por sua vez, são responsáveis por usar os métodos de acesso (*e.g.,* protocolos, APIs, bibliotecas) necessários para se comunicar com os recursos adaptados e extrair deles os resultados requisitados pela camada superior. Além disso, os *wrappers* são responsáveis por converter os dados obtidos dos recursos adaptados para um formato de dados canônico conhecido pelo sistema de *mashups*.

A tarefa de adaptação apresenta alta complexidade, devido à heterogeneidade tanto dos métodos de acesso quanto dos formatos de dados utilizados pelos recursos externos. O modelo de *wrappers* lida com essa complexidade permitindo que cada adaptador trate de forma única as características de acesso do recurso adaptado (THOR; AUMUELLER; RAHM, 2007). Como os *wrappers* respondem ao sistema através de uma interface comum, os métodos de acesso e os protocolos envolvidos na interação *wrapper*-recurso são transparentes para o sistema de *mashups*. A arquitetura do sistema de *mashups* de gerenciamento assume, portanto, que cada *wrapper* é capaz de interagir com o recurso adaptado e responder ao sistema de *mashups* através de uma interface conhecida pelo mesmo. A discussão em detalhes de como os cada *wrapper* adapta seu respectivo recurso foge ao escopo dessa dissertação.

Os *wrappers* podem ser classificados de acordo com os recursos que eles adaptam. Alguns exemplos incluem:

- **Wrappers de protocolo** adaptam protocolos distintos ao sistema de *mashups*, permitindo que as composições utilizem recursos acessíveis através desses protocolos. Por exemplo, um *wrapper* para o protocolo SOAP permite que um administrador utilize Web Services nos seus *mashups* de gerenciamento. Um *wrapper* para SNMP, por sua vez, permite que elementos gerenciados via SNMP sejam acessados por esses *mashups*;

- **Wrappers de dados** servem para possibilitar que o sistema de *mashups* compreenda e suporte diferentes formatos de dados. Alguns exemplos de formatos bastante utilizados no gerenciamento de redes incluem CSV, XML e formatos de fluxos de IP (*NetFlow*);

- **Wrappers de ferramentas** são responsáveis por se comunicar com ferramentas de gerenciamento e adaptá-las para o sistema de *mashups*. Incluem, por exemplo, *wrappers* que extraem informações de interfaces Web de ferramentas como MRTG e NTOP;

- **Wrappers de APIs *online*** adaptam recursos Web disponíveis através de APIs *online* como, por exemplo, serviços de mapas e geolocalização.

O sistema de *mashups* de gerenciamento conhece seus *wrappers* através de descrições dos mesmos, que são armazenadas no **Repositório de *Wrappers***, localizado na camada de composição. O objetivo dessas descrições é permitir que o sistema enxergue os recursos externos acessíveis de uma forma unificada. Para tal, as informações contidas nas descrições contém, no mínimo, especificações das operações que os *wrappers* podem realizar nos recursos adaptados, dos dados de entrada necessários e dos dados de saída retornados. Essas são as informações essenciais para descrever um *wrapper*, entretanto, informações

adicionais podem ser incluídas para permitir que o sistema conheça mais detalhes sobre os recursos adaptados. Por exemplo, muitas APIs *online* exigem credenciais de usuário, somente respondendo requisições mediante a submissão de uma chave de acesso denominada *developer key*. Essa chave pode fazer parte da descrição do *wrapper*, eliminando a necessidade de cada usuário do sistema ter também de se cadastrar no provedor da API *online*.

A partir das descrições dos *wrappers*, a camada de apresentação deve ser capaz de criar representações visuais dos mesmos que, na perspectiva do desenvolvedor de *mashups*, correspondem aos recursos externos. A camada de composição, por sua vez, utiliza essas descrições para validar o uso dos *wrappers* durante a criação das composições e para invocar os mesmos durante a execução dos *mashups*. A próxima seção abordará em mais detalhes essa camada de composição, apresentando os componentes que possibilitam a integração dos recursos externos adaptados pelos *wrappers*.

3.2 Camada de Composição

A camada de composição é o núcleo do sistema de *mashups*. Seu objetivo é possibilitar a composição de recursos que dará origem aos *mashups* de gerenciamento (LIU et al., 2007). Para cumprir esse objetivo, essa camada é responsável pelas etapas envolvidas tanto na composição de recursos quanto na publicação dos *mashups*. Essas responsabilidades são desempenhadas, respectivamente, pelos seguintes componentes da camada: ***Composer*** e ***Publisher***. Uma representação esquemática da camada pode ser vista na figura 3.3.

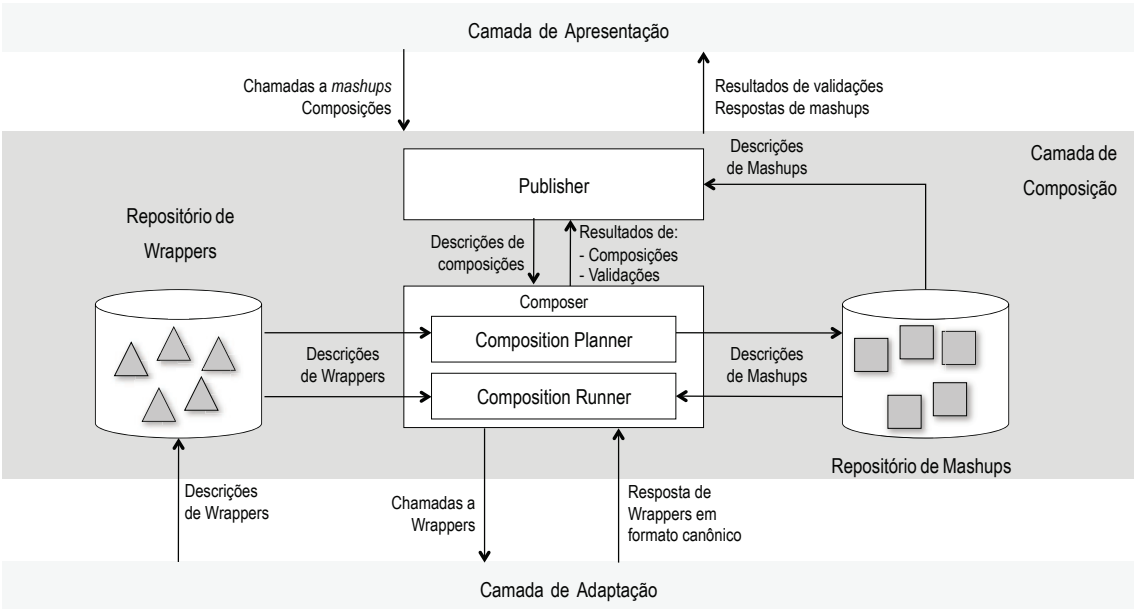

Figura 3.3: A camada de composição

O *composer* é o componente da camada de composição responsável pelas etapas de **planejamento** e **execução** dos *mashups*. Essas responsabilidades são centrais ao funcionamento do sistema de *mashups*, o que faz desse componente o mais importante da camada. O *composer* é dividido em dois módulos: ***Composition Planner***, responsável

pelo planejamento de *mashups* e ***Composition Runner***, responsável pela execução dos mesmos. Essa divisão em módulos é necessária pois essas etapas se relacionam de forma distinta tanto com os demais componentes da arquitetura quanto com os diferentes usuários do sistema de *mashups* de gerenciamento.

O objetivo do *composition planner* é garantir a integridade das composições criadas no sistema. A etapa de planejamento dos *mashups* (*mashup planning*) é a forma utilizada por ele para cumprir esse objetivo. O *planning* consiste na validação das composições definidas pelo desenvolvedor de *mashups* e no retorno do resultado dessa validação. Quando essa validação falha (*i.e.*, quando a composição não é consistente), é responsabilidade do *planner* dar indícios dos motivos dessa falha, os quais serão usados pelo sistema para dar ***feedback*** ao desenvolvedor sobre o seu *mashup*. Esse tipo *feedback* é de enorme importância para a usabilidade do sistema de *mashups* (NIELSEN, 1994). Além disso, o *planner* é um módulo importante para a arquitetura por permitir que os demais componentes da mesma trabalhem assumindo os *mashups* existentes atendem premissas de integridade garantidas pela validação realizada no *planning*.

O *composition runner*, por sua vez, é o módulo do *composer* responsável por executar os *mashups* existentes no sistema. Para tal, esse módulo comporta-se como uma *engine* de execução capaz de receber a descrição de uma composição de recursos, executar a lógica definida nessa descrição invocando os componentes (*e.g.*, *wrappers*) necessários e, por fim, retornar o resultado dessa execução. Esse retorno pode ser realizado tanto para a camada de apresentação, caso o *mashup* tenha sido invocado por um administrador, quanto para o próprio *composer*, caso o *mashup* executado seja componente de uma outra composição mais sofisticada.

Assim como os *wrappers*, os *mashups* existem no sistema através de descrições armazenadas em um repositório específico, denominado **Repositório de *Mashups***. Essas descrições incluem, no mínimo, informações sobre as entradas e saídas da composição, bem como a lógica definida pelo desenvolvedor. Além de permitir que o *composition runner* execute os *mashups*, como já visto, as descrições armazenadas no repositório podem ser usadas na etapa de *planejamento*, caso o *mashup* descrito seja usado em uma composição mais sofisticada. A implementação do sistema de *mashups* pode optar por permitir o armazenamento de descrições não validadas caso se deseje que o administrador possa salvar composições não finalizadas. Entretanto, tanto no planejamento quanto na execução de *mashups*, o *composer* trabalha apenas com composições já validadas.

O outro componente da camada de composição, denominado ***Publisher***, é responsável por controlar a publicação dos *mashups* através de permissões de acesso que definem, por exemplo, quais *mashups* um determinado administrador de redes pode executar, modificar ou reutilizar. Essas permissões podem ser definidas tanto pelo desenvolvedor do *mashup* quanto pelo responsável por administrar o sistema. Pode-se definir também permissões de acesso para os *wrappers*, o que permite estabelecer quais recursos externos estarão acessíveis para um determinado administrador de redes. Esse controle de permissões, tanto de *wrappers* quanto de *mashups*, permite que o sistema de *mashups* de gerenciamento de redes garanta o cumprimento de regras e limitações administrativas às quais os administradores estão sujeitos. Por exemplo, caso a administração da rede seja dividida em sub-domínios independentes, permissões de acesso possibilitam que sejam definidos *wrappers* e *mashups* de gerenciamento exclusivos para os administradores de um determinado domínio (SANTOS et al., 2011).

Além do controle de permissões, o *publisher* tem outras responsabilidades. Uma delas é o controle de usuários do sistema, ou seja, manter as credenciais (*e.g.*, *login* e senha)

dos administradores registrados no sistema de *mashups* e autenticar o acesso dos mesmos. Além desse controle, o compartilhamento de composições entre diferentes sistemas de *mashups* de gerenciamento é também responsabilidade do *publisher*. O objetivo desse compartilhamento é possibilitar um cenário onde um *mashup* é criado em um sistema e executado em outro, permitindo que administradores de diferentes redes compartilhem *mashups* (BEZERRA et al., 2010). Para alcançar esse objetivo, os *publishers* dos diferentes sistemas de *mashups* comunicam-se, trocando descrições das composições compartilhadas, as regras de acesso a essas composições e, possivelmente, *wrappers* utilizados nas mesmas. Ao contrário, por exemplo, da validação realizada pelo *planner*, o compartilhamento descrito não é uma necessidade do sistema de *mashups*. Ele é, no entanto, uma forma de auxiliar o administrador de redes em questões de gerenciamento inter-domínio possibilitando, por exemplo, que administradores de diferentes redes troquem *mashups* de gerenciamento que expõem informações relevantes das suas redes. Além disso, o compartilhamento entre diferentes sistemas de *mashups* de gerenciamento possibilita o surgimento de um ecossistema de *mashups* de gerenciamento em nível global, caso esse compartilhamento seja adotado em diversos domínios administrativos na Internet.

Para cumprir as responsabilidades definidas acima, o *publisher* atua como intermediário na comunicação entre a camada de composição e as interfaces com o usuário do sistema de *mashups*. A camada de apresentação, que contém essas interfaces, será abordada na seção seguinte.

3.3 Camada de Apresentação

A camada superior da arquitetura, denominada camada de apresentação, é a responsável pela interação entre o sistema de *mashups* de gerenciamento e seus usuários (*i.e.*, administradores de redes). O principal objetivo dessa camada é possibilitar que administradores de redes criem *mashups* de gerenciamento independentemente de habilidade em desenvolvimento de *software*. Além disso, ela deve permitir que esses administradores encontrem e utilizem *mashups* de gerenciamento já existentes no sistema tanto para atender necessidades da rede ou como componentes em composições mais softisticadas. A importância da camada de apresentação é evidenciada pela própria proposta dos *mashups* como tecnologia centrada no usuário final (MERRIL, 2003).

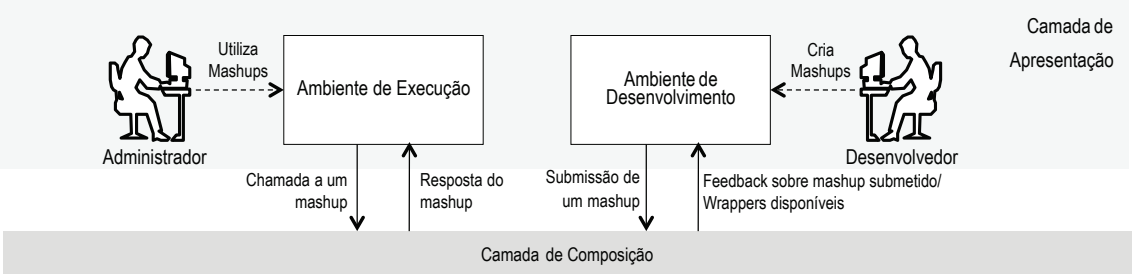

Figura 3.4: A camada de apresentação

De forma a melhor atender às necessidades dos dois tipos de usuários definidos para a arquitetura, a camada de apresentação é subdividida em dois ambientes (YU et al., 2008), como é possível ver na figura 3.4. O **Ambiente de Desenvolvimento** governa a interação com o desenvolvedor de *mashups*, possibilitando que o mesmo componha *mashups* de

gerenciamento. O **Ambiente de Execução**, por sua vez, é responsável por interagir com o administrador de redes interessado apenas em utilizar composições existentes no sistema. Os dois ambientes serão detalhados no restante da seção.

O objetivo do ambiente de desenvolvimento é permitir que o desenvolvedor crie *mashups* de gerenciamento através da composição de recursos existentes tanto na rede gerenciada quanto na Internet, sem que para tal esse desenvolvedor precise de habilidade em desenvolvimento de *software*. Para cumprir esse objetivo, o ambiente de desenvolvimento emprega uma **Interface de Composição**. Ao interagir com essa interface, o desenvolvedor deve ser capaz de definir a lógica da sua composição através da manipulação de **Elementos de Interação**. Tais elementos consistem em abstrações de alto nível para os componentes de um *mashup*, que podem ser tanto recursos internos do sistema (*e.g.*, operadores sobre dados, *mashups* existentes) quanto recursos externos adaptados por *wrappers*. Os elementos de interação referentes aos operadores do sistema fazem parte da implementação do mesmo. Já no caso de *wrappers* e *mashups* utilizados como componentes, a interface de composição utiliza as descrições desses componentes para construir os elementos de interação apresentados para o usuário.

O objetivo do ambiente de execução, por sua vez, é permitir que os administradores de rede recuperem e utilizem *mashups* de gerenciamento existentes para atender suas necessidades. Para tal, esse ambiente deve conter uma interface de busca de *mashups*. Além disso, ele deve ser capaz de exibir o resultado de composições invocadas pelo usuário, os quais são, normalmente, páginas Web. O ambiente de execução também pode permitir que determinadas composições sejam acessadas através de endereços únicos (*i.e.*, *Uniform Resource Locator* - URL). Dessa forma, administradores podem utilizar *mashups* de gerenciamento como aplicações Web *standalone*, sem que eles tenham que interagir diretamente com a interface de busca sistema de *mashups*. Nesse caso, é responsabilidade do ambiente de execução processar requisições às URLs dos *mashups* e apresentar o resultado destes ao usuário.

Foram abordados, até então, todos os componentes da arquitetura proposta para um sistema de *mashups* de gerenciamento, desde os adaptadores de recursos externos até as interfaces com o usuário. A próxima seção descreverá como esses componentes interagem na criação e na execução de *mashups*.

3.4 Interação Entre os Componentes da Arquitetura

Essa seção apresenta em detalhes o fluxo de interação entre os componentes da arquitetura de sistemas de *mashups* de gerenciamento de redes durante as atividades de criação e execução de *mashups*. Os passos envolvidos nessas atividades e os componentes da arquitetura envolvidos estão esquematizados na figura 3.5 e serão descritos em detalhes no decorrer da seção.

1. A camada de adaptação popula o repositório de *wrappers* com descrições dos adaptadores disponíveis. Todo o fluxo de interação apresentado parte da premissa que esse repositório estará populado, ou seja, que há adaptadores disponíveis para os recursos externos utilizados nos *mashups*;

2. O desenvolvedor submete *um mashup* criado no ambiente de desenvolvimento para a camada de composição. Nesse passo, a camada de apresentação envia uma descrição da lógica da composição desenvolvida para o *publisher*, dando início à etapa de planejamento do *mashup*. O publisher processará essa descrição verificando, por

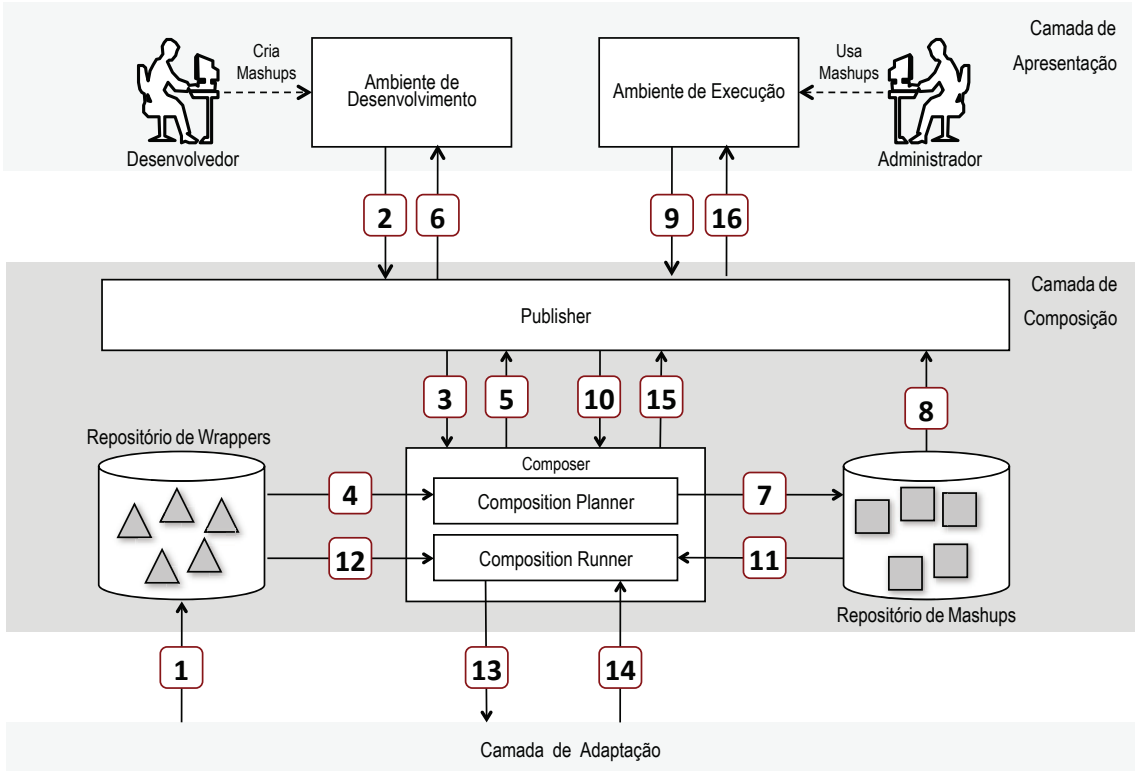

Figura 3.5: Representação esquemática da interação entre os componentes da arquitetura.

exemplo, se o usuário tem as permissões adequadas para utilizar os componentes do *mashup* submetido;

3. Após processar a composição submetida, o *publisher* envia a descrição da mesma para que o *composer* realize a validação. Essa validação é realizada pelo módulo do composer denominado *composition planner*. O que é validado nela depende largamente da implementação do sistema de *mashups*. No mínimo, é importante que o *compostion planner* garanta que os componentes do *mashup* estão recebendo entradas adequadas e que suas saídas são devidamente tratadas. Outras verificações que o *planner* pode fazer incluem, por exemplo, verificar se a lógica de composição apresenta laços lógicos infinitos ou verificar se os recursos externos utilizados ainda estão disponíveis;

4. O *planner* recupera as descrições dos *wrappers* utilizados no *mashup* sendo validado e extrai delas as informações sobre os mesmos que são necessárias para a validação (*e.g.*, entradas, saídas, operações disponíveis);

5. Ao fim da validação, o resultado dela é retornado ao *publisher*. Essa validação é considerada bem sucedida se nenhuma inconsistência for encontrada na composição. Nesse caso, o resultado retornado informa que o *mashup* é válido. Já no caso de haver inconsistências no *mashup*, a validação é considerada mal-sucedida. O resultado retornado nesse caso informa que o *mashup* submetido é inválido e quais os motivos que impediram o *planner* de concluir a validação com sucesso (*e.g.*, quais verificações realizadas falharam);

6. Tendo recebido o resultado da validação, o *publisher* retorna esse resultado para a camada de apresentação. Nela, o ambiente de desenvolvimento apresenta o resul-

tado para o desenvolvedor de *mashups*. Caso a validação tenha sido mal-sucedida, o fluxo de interação retorna para o passo 2, após o desenvolvedor realizar ajustes na sua composição;

7. Já no caso da validação ter sido bem sucedida, o *composition planner* armazena a descrição válida da composição no repositório de *mashups*. Ao fim desse passo, a criação do *mashup* está concluída com sucesso e é encerrada a etapa de planejamento;

8. O *publisher* publica a descrição do *mashup* que agora existe no repositório. Esse *mashup* pode, então, ser encontrado pela interface de busca da camada de apresentação;

9. No ambiente de execução, o administrador solicita a execução da composição criada. O ambiente de execução, por sua vez, envia uma *chamada* ao *mashup* para a camada de composição, a qual é recebida pelo *publisher*;

10. Após verificar que o usuário tem as permissões necessárias para acessar o *mashup* invocado, o *publisher* solicita que o *composer* execute tal *mashup*. Para atender essa solicitação, o *composer* aciona o *composition runner*;

11. O *composition runner* recupera a descrição do *mashup* solicitado, contida no repositório de *mashups*. De posse dessa descrição, o *runner* vai executar a lógica da composição, orquestrando chamadas aos *wrappers* dos recursos externos utilizados na composição e realizando operações sobre os resultados retornados por esses *wrappers*;

12. Para interagir com os *wrappers*, o *composition runner* recupera as descrições dos mesmos contidas no repositório de *wrappers*;

13. Baseado partir nas informações contidas nessas descrições (*e.g.*, tipos de entrada aceitos pelo adaptador, operações disponíveis), o *composition runner* realiza as chamadas aos *wrappers* da camada de adaptação. Ao receber essa chamada, cada *wrapper* comunica-se com o recurso externo adaptado, realiza as operações solicitadas e converte o resultado dessas operações para o formato padronizado conhecido pelo sistema de *mashups*;

14. O *wrapper*, então, retorna o resultado da comunicação com o recurso externo para o composition *runner*;

15. Após se comunicar com os *wrappers* necessários e executar as operações definidas pela lógica da composição, o *composition runner* retorna o resultado do *mashup* (normalmente uma página Web) para o *publisher*;

16. O *publisher* encaminha esse resultado para o ambiente de execução, o qual irá finalmente exibi-lo para o usuário, concluindo a etapa de execução do *mashup*.

O fluxo descrito corresponde a um exemplo de funcionamento básico do sistema de *mashups* de gerenciamento. Ele não descreve, no entanto, algumas situações mais complexas desse funcionamento. Por exemplo, o reuso de *mashups* existentes no sistema como componentes em outras composições (*i.e.*, composições compostas) não é abordado

pelo fluxo descrito acima. No planejamento de uma composição composta, o *planner* utiliza as descrições dos *mashups* utilizados como componentes para realizar a validação, de forma análoga ao que ocorre com os *wrappers* (passo 4). Já na execução, o *runner* faz uso dessas descrições para executar os *mashups* reutilizados, ou seja, a lógica de composições compostas inclui a lógica dos seus *mashups* componentes. O motivo de não incluir as interações envolvidas no reuso de *mashups* no fluxo de interação apresentado é didático, já que seria necessário incluir mais passos e criar ciclos lógicos para descrever visualmente uma interação que, como visto, é análoga à utilização de *wrappers*.

As etapas de planejamento e execução de *mashups* descritas nessa seção são realizadas em momentos diferentes da utilização do sistema. Elas foram representadas de forma sequencial na figura 3.5 e descritas dessa forma pois a etapa de execução necessita que exista pelo menos um *mashup* salvo no repositório, ou seja, é requisito para a etapa de execução que pelo menos uma etapa de planejamento tenha sido efetuada. Apesar da descrição sequencial apresentada, em um cenário de uso real do sistema, a criação de *mashups* geralmente segue o modelo de desenvolvimento de *software* denominado prototipação evolucionária (SOMMERVILLE, 2004). Segundo esse modelo, o desenvolvedor irá solicitar diversas validações e execuções de composições parciais (*i.e.*, protótipos) até que seu *mashup* se comporte da forma desejada. A principal particularidade da criação de *mashups* em relação ao modelo de prototipação reside no fato de que o próprio desenvolvedor define os requisitos da aplicação prototipada e valida os protótipos, visto que ele também é o usuário final do *mashup* sendo desenvolvido.

Essa seção demonstrou as interações entre os componentes da arquitetura durante a criação e a execução de *mashups* de gerenciamento. A próxima seção abordará em maiores detalhes os elementos de interação utilizados pelo desenvolvedor durante essa criação, visando permitir um entendimento mais aprofundado de como *mashups* de gerenciamento são criados.

3.5 Elementos de Interação

Como foi visto anteriormente, no processo de criação de *mashups* de gerenciamento o desenvolvedor manipula **elementos de interação** para definir a lógica da sua composição. Esses elementos de interação são abstrações para as fontes de dados e recursos utilizados na composição do *mashup*. Essas abstrações podem ser materializadas, por exemplo, em blocos visuais com os quais o desenvolvedor pode interagir no ambiente de desenvolvimento do sistema. Os elementos de interação podem ser classificados de acordo com sua função na criação de *mashups*. A seguir, são descritos os principais tipos de elementos de interação conhecidos, entretanto, a arquitetura é flexível o suficiente para suportar a definição de outros tipos:

- **Visual**: Elementos de interação do tipo visual representam as formas que os resultados das composições podem ser apresentadas para o usuário. Eles incluem, por exemplo, elementos que criam mapas interativos, árvores, tabelas ou fluxogramas;

- **Controle**: Esses elementos controlam o fluxo de execução da composição, representando conceitos de lógica de programação em formato abstrato compreensível para usuários leigos. Eles permitem que o desenvolvedor de *mashups* defina, por exemplo, condicionais lógicos e laços iterativos;

- **Operação**: Esse tipo de elemento permite a realização de operações sobre os dados

recuperados dos recursos externos. Incluem, por exemplo, filtros, ferramentas de *merge* automatizado de dados, operações matemáticas e *booleanas*;

- **Adaptação**: Elementos de adaptação representam os recursos externos utilizados em *mashups* de gerenciamento. Eles são gerados pelo ambiente de desenvolvimento com base nas descrições dos *wrappers*;

- **Reuso**: Esse tipo de elemento representa *mashups* existentes no sistema que podem ser utilizados como blocos de composição em outros *mashups*, permitindo a criação de composições composições compostas. De forma análoga aos elementos de adaptação, elementos de interação do tipo reuso são gerados a partir das descrições de composições existentes no repositório de *mashups*;

- ***Daemon***: Um elemento do tipo *daemon* representa um *mashup* que será continuamente executado em *background* pelo *sistema*. A definição desse tipo de elemento permite a criação de *mashups* dedicados, por exemplo, à monitoração contínua de tráfego na rede. Entretanto a execução de diversos *daemons* pode ter implicações de desempenho para o sistema de *mashups* cuja investigação foge ao escopo dessa dissertação;

- ***Scripting***: Esses elementos representam *scripts* definidos por administradores com habilidade em desenvolvimento de *software*, permitindo que tais administradores empreguem essas habilidades na criação de *mashups* de gerenciamento mais complexos. Elementos de *scripting* permitem que a expressividade sistema de *mashups* escale de acordo com a experiência do administrador em desenvolvimento. Entretanto, podem existir implicações complexas de estabilidade e segurança em permitir que *mashups* de gerenciamento executem código arbitrário. A investigação dessas implicações foge ao escopo dessa dissertação;

- **Confidencialidade de dados**: Elementos de confidencialidade são responsáveis por identificar informações sensíveis e/ou confidenciais nas composições desenvolvidas (SANTOS et al., 2011). Na criação de *mashups*, os *desenvolvedores* podem empregar elementos desse tipo para especificar a acessibilidade das informações contidas na sua composição. Já na execução, esses elementos são utilizados pelo sistema para definir quais informações do *mashup* serão exibidas para o usuário.

A arquitetura para um sistema de *mashups* de gerenciamento de redes foi apresentada e discutida em profundidade nesse capítulo. O próximo passo desse trabalho é avaliar como os conceitos propostos por essa arquitetura se comportam na prática. Para tal, um protótipo de sistema de *mashups* de gerenciamento denominado *Network Management Mashup System* (NMMS) será apresentado e discutido na próxima seção.

4 IMPLEMENTAÇÃO - *NETWORK MANAGEMENT MASHUP SYSTEM*

Para viabilizar a observação, na prática, da aplicação de *mashups* no gerenciamento de redes, foi criado um protótipo de sistema de *mashups* de gerenciamento denominado *Network Management Mashup System* (NMMS). Esse protótipo é baseado nos princípios propostos pela arquitetura descrita no capítulo anterior, funcionando efetivamente como uma prova de conceito da mesma. Dado que a principal funcionalidade do sistema descrito pela arquitetura é possibilitar a criação e a execução de *mashups* de gerenciamento, decidiu-se por não implementar, no protótipo, requisitos não funcionais como, por exemplo, o gerenciamento de usuários e o compartilhamento de *mashups* com outros sistemas. A decisão de não atender a requisitos não funcionais como esses permitiu uma implementação mais simples e ágil dos componentes fundamentais da arquitetura.

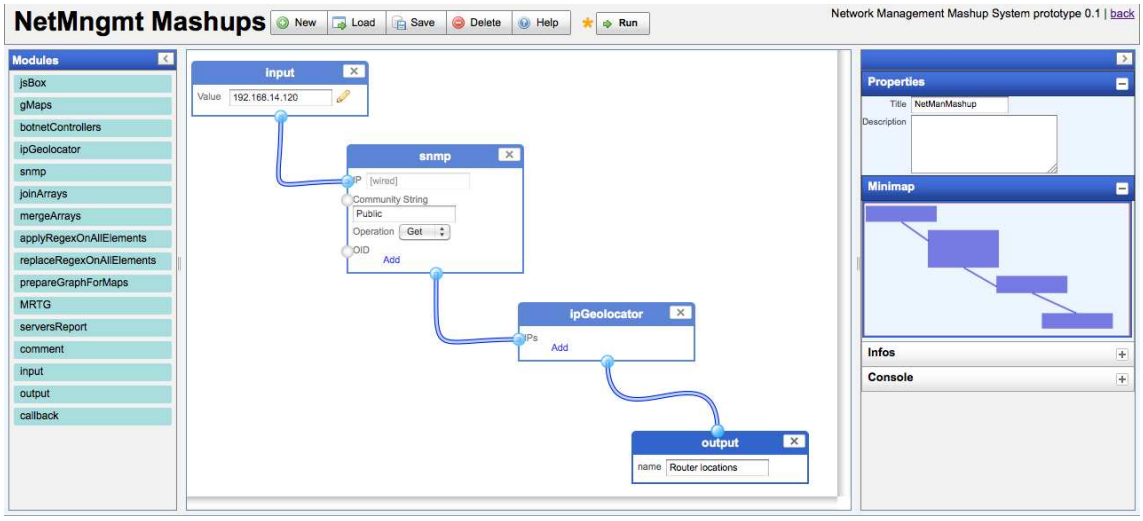

Figura 4.1: O *Network Management Mashup System*

O NMMS é um sistema Web que permite a criação de *mashups* de gerenciamento de redes através de uma interface visual de composição. Essa interface, que pode ser vista na figura 4.1, é baseada na manipulação e interconexão de blocos funcionais lógicos (*i.e.*, elementos de interação) que abstraem recursos disponíveis na Web (*e.g.*, serviços de geolocalização), ferramentas de gerenciamento (*e.g.*, MRTG, SNMP) e operadores do próprio sistema de *mashups* (*e.g.*, saída do *mashup*, *merge* de dados). No NMMS, o administrador pode executar suas composições em tempo real e receber *feedback* imediato

sobre elas. Além disso, os *mashups* criados no NMMS podem ser publicados na Web através de URLs disponibilizadas pelo sistema.

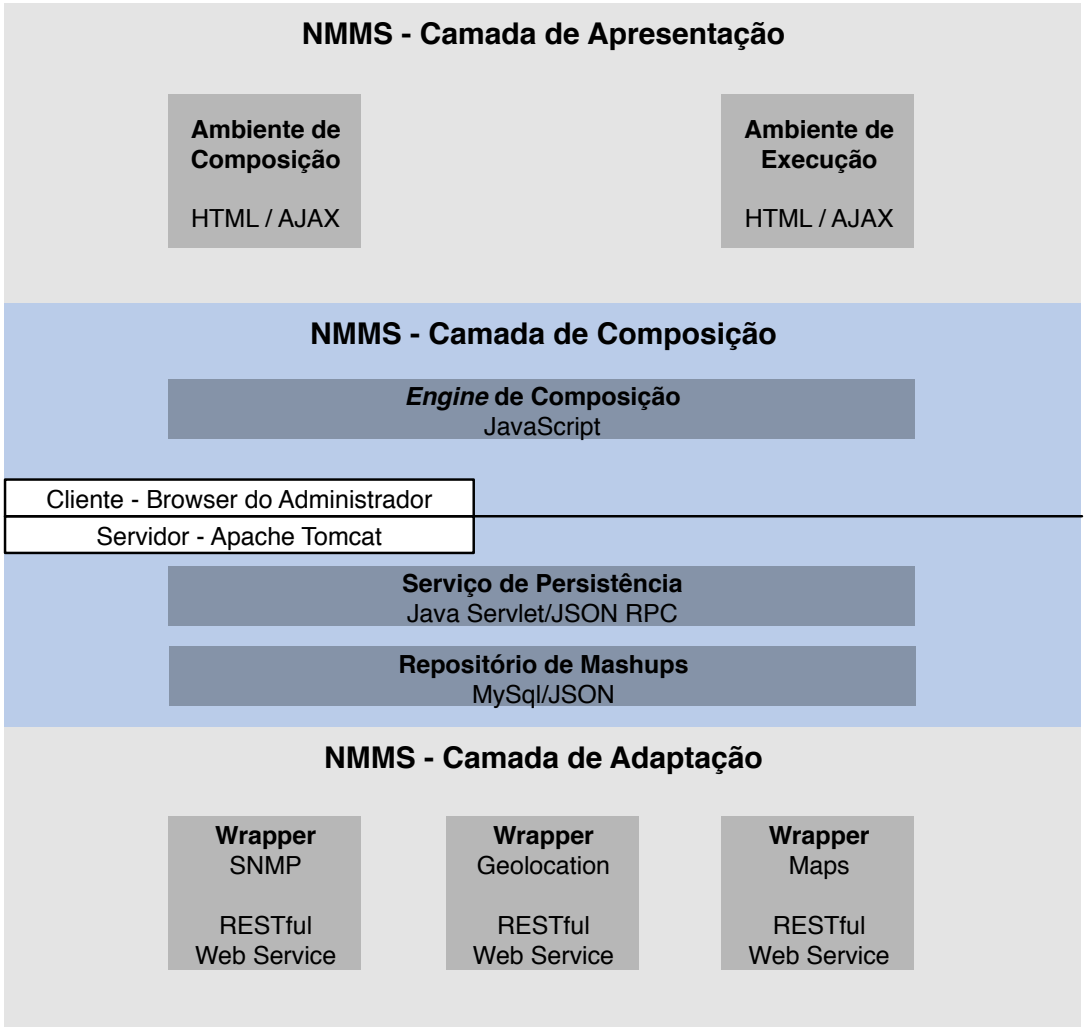

Figura 4.2: Arquitetura do *Network Management Mashup System - Panorama tecnológico*

Como pode ser visto na figura 4.2, o NMMS é uma aplicação Web baseada em AJAX que utiliza JSON como formato de intercâmbio de dados. Como a maioria dos sistemas AJAX (GARRET, 2005), o lado do cliente do sistema é baseado em HTML e JavaScript, sendo executado no *browser* do usuário. Esse cliente inclui tanto as interfaces do usuário na camada de apresentação, implementadas utilizando HTML, *Cascading Style Sheets* (CSS) e JavaScript, quanto a engine de composição do sistema, desenvolvida em JavaScript.

O lado do servidor do NMMS, por sua vez, foi implementado utilizando tecnologias Java como Servlets e Java *Server Pages* (JSP), como mostra também a figura 4.2. Ele é executado no servidor de aplicação *Apache Tomcat*[1]. No lado do servidor está localizado o repositório do sistema, que consiste em um banco de dados relacional MySql[2]. Nesse

[1]http://tomcat.apache.org/
[2]http://www.mysql.com/

repositório, são armazenadas descrições dos *mashups* criados no sistema, que são codificadas em JSON. A comunicação com o repositório é realizada através de um serviço de persistência implementado através da tecnologia Java Servlets. Finalmente, a camada de adaptação do NMMS também está localizada no lado do servidor do sistema. Os *wrappers* dessa camada são implementados como RESTful Web Services e respondem a chamadas HTTP feitas pela *engine* de composição.

As decisões arquiteturais e tecnológicas envolvidas na escolha e utilização das tecnologias apresentadas serão discutidas em detalhes nas próximas seções. Cada seção trata de uma das camadas do NMMS, seguindo a mesma lógica *bottom up* adotada no capítulo anterior para a descrição da arquitetura.

4.1 NMMS - Camada de Adaptação

A camada de adaptação do NMMS está implementada no lado do servidor do sistema e é responsável por armazenar e gerenciar os *wrappers* (*i.e.*, adaptadores para recursos externos) do mesmo. Como pode ser visto na figura 4.3, a implementação dos *wrappers* do NMMS é baseada em RESTful Web Services (PAUTASSO; ZIMMERMANN; LEYMANN, 2008). O NMMS realiza chamadas aos *wrappers* de forma padronizada através de requisições GET e POST do protocolo HTTP. Foi definido na implementação do protótipo que cada *wrapper* deve ter uma única operação denominada ***Run***, a qual recebe dados de entrada, comunica-se com o recurso adaptado, converte o resultado dessa comunicação para o formato padronizado pelo NMMS e retorna esse resultado para as camadas superiores do sistema de *mashups*. O formato utilizado como padrão pelo NMMS é o JSON.

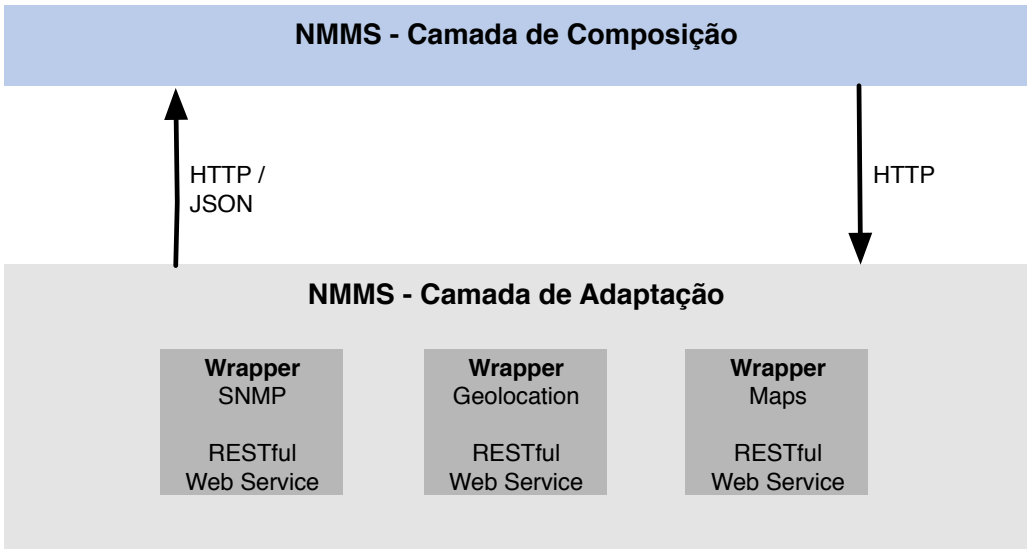

Figura 4.3: A camada de adaptação do *Network Management Mashup System*

O *wrapper* SNMP é um exemplo de adaptador do NMMS, o qual permite que o sistema de *mashups* se comunique com elementos gerenciados na rede através do protocolo SNMP. Para tal, esse adaptador emprega um *gateway* SNMP - Web Services em nível de protocolo (VIANNA et al., 2007), que converte chamadas ao *wrapper* para operações do protocolo SNMP e comunica-se com os agentes dos elementos gerenciados para realizar

essas operações. Como o modelo de *wrappers* do NMMS só prevê uma operação por adaptador (*i.e., run*), o suporte a múltiplas operações do SNMP (*e.g.,* GET, WALK) foi implementado através de um parâmetro adicional denominado *operation*, que informa ao *wrapper* qual operação SNMP ele deve executar. A seguir, são apresentadas as entradas e saídas do adaptador SNMP e, em seguida, a figura 4.4 mostra a descrição do *wrapper* armazenada pelo NMMS:

- **Entradas**:

  IP - Endereço IP do agente SNMP contactado;

  Community String - *String* de comunidade de acesso ao agente SNMP;

  Operation - Operação SNMP realizada;

  OID - Lista de identificadores (Object Identifier - OID) dos objetos SNMP solicitados;

- **Saída**:

  Valores dos objetos solicitados.

```
"name": "snmp",
"container": {
    "xtype":"WireIt.FormContainer",
    "title": "SNMP",
    "fields": [{
        "type": "string",
        "inputParams": {"label":"IP", "name":"IP", "wirable":true}
      }, {
        "type": "string",
        "inputParams": {"label":"Community String", "name":"Community String", "wirable":true}
      }, {
        "type": "select",
        "inputParams": {"label":"Operation", "name":"operation", "selectValues": ["Get", "Walk"]}
      }, {
        "type": "list",
        "inputParams": {"label":"OID", "name":"OID","wirable": true}
    }],
    "terminals": [{
        "name": "out",
        "direction": [0, 1],
        "offsetPosition": {"left":86, "bottom":-15 },
        "ddConfig": {"type":"output", "allowedTypes":["input"]}
    }]
}
```

Figura 4.4: Descrição do *Wrapper* SNMP

No lado do servidor do NMMS são armazenadas as descrições de cada *wrapper* existente no sistema. Um exemplo de descrição pode ser visto na figura 4.4. Tais descrições seguem o formato definido pela biblioteca WireIt (ABOUAF, 2007). As informações contida nessas descrições são o nome do *wrapper*, nomes e formatos das suas entradas e saídas. Dado que cada adaptador só conta com uma operação (*i.e., run*), informações referentes a operações de *wrappers* são desnecessárias.

As informações contidas na descrição dos *wrappers* consistem no conjunto mínimo necessário para que a camada de composição do NMMS valide composições que utilizam esses adaptadores e os invoque durante a execução dessas composições. Tais informações também são utilizadas pela camada de apresentação para gerar o elemento de interação referente ao *wrapper* descrito. Essas camadas serão abordadas em detalhes nas seções seguintes.

4.2 NMMS - Camada de Composição

A camada de composição do NMMS é responsável por validar os *mashups* sendo criados, armazenar essas composições e executá-las. Como pode ser visto na figura 4.5, essa camada contém componentes tanto do lado do cliente quando do lado do servidor do NMMS. Os componentes do lado do servidor são responsáveis pelo armazenamento dos *mashups*. Já no lado do cliente está implementada a *engine* de composição, componente que realiza tanto a validação quanto a execução dos *mashups*. A decisão de implementar a *engine* de composição no lado do cliente propiciou maior facilidade de desenvolvimento, devido ao uso da biblioteca WireIt (ABOUAF, 2007) para execução de *workflows* e à possibilidade de comunicação dessa *engine* com a camada de apresentação através do uso de métodos da linguagem JavaScript.

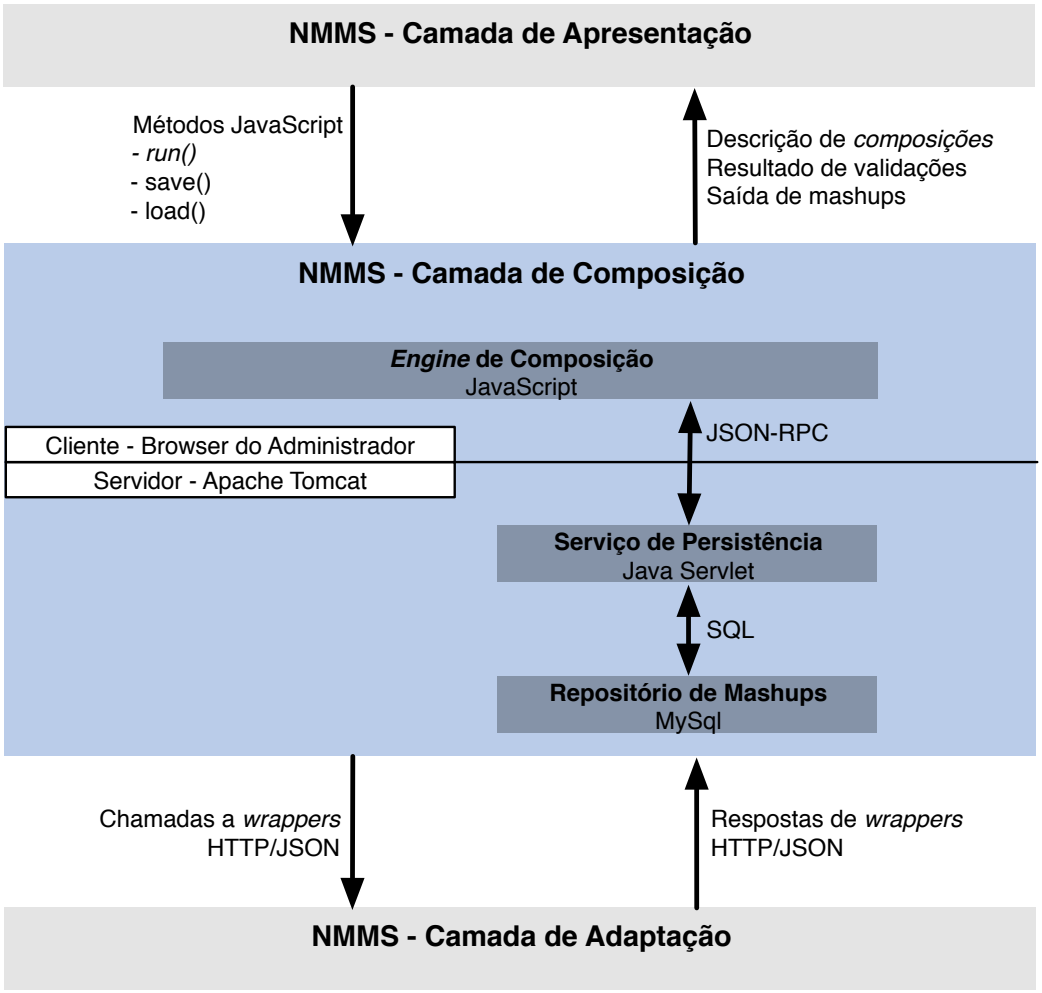

Figura 4.5: A camada de composição do *Network Management Mashup System*

As composições no NMMS consistem em *workflows* definidos através da interconexão de componentes funcionais denominados **módulos**. Cada módulo recebe *N* entradas, executa uma operação (denominada *run*) sobre essas entradas e retorna N saídas para o próximo componente do *workflow*. Uma composição também é considerada um módulo, onde suas entradas e saídas são definidas pelo desenvolvedor de *mashups* e a operação *run* é a lógica definida pelo mesmo. Ou seja, o NMMS suporta a existência de compo-

sições que utilizam outros *mashups* como módulos nos seu *workflow* (*i.e.,* composições compostas).

Na etapa de *composition planning* do NMMS, a *engine* de composição valida o *mashup* sendo criado, verificando se não há entradas indevidamente vazias ou conexões de tipos incompatíveis entre dois módulos no *workflow*. Essa verificação de tipos leva em consideração tipos arbitrários que podem ser definidos tanto por *wrappers* como por desenvolvedores de *mashups*. Ao final de uma validação bem sucedida, a composição validada é salva no repositório de *mashups* e pode ser executada pelo NMMS.

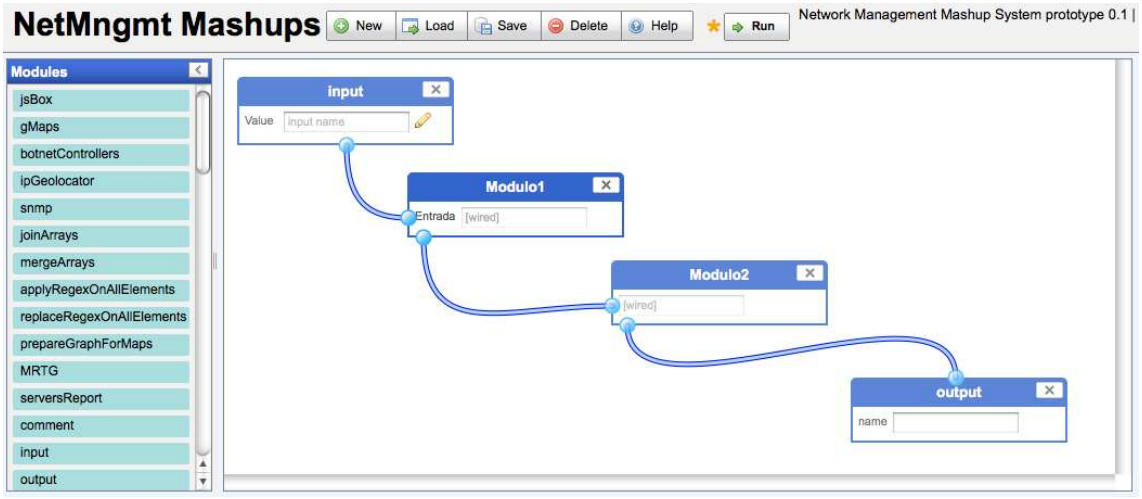

Figura 4.6: Exemplo de *workflow* composto no NMMS

A execução das composições validadas é realizada a partir de descrições de *mashups* salvas no repositório. Nessa execução, o NMMS assume que o ponto de partida do *workflow* é o módulo conectado ao componente interno do sistema denominado *input*, utilizado para definir as entradas do *mashup*. No exemplo de *workflow* mostrado na figura 4.6, pode-se observar portanto que o ponto de partida é o módulo denominado *Modulo1*. A partir desse ponto de partida, cada módulo recebe como entrada as saídas do módulo anterior, executa a operação *run* sobre essas entradas e returna o resultado dessa operação para o próximo *módulo* do fluxo. Quando o módulo referente a um *wrapper* é executado, a *engine* invoca esse *wrapper* na camada de adaptação e o mesmo responde com o resultado da comunicação com o recurso externo adaptado. Já no caso do módulo executado ser uma composição salva no NMMS, ela é executada pela *engine* de composição e suas saídas são passadas para o próximo módulo. A execução de um *mashup* termina quando o último componente é executado. Esse componente deve estar ligado ao módulo interno *output*, que define a saída do *mashup*. No exemplo da figura 4.6, o módulo *Modulo2* está conectado a *output* e é o último a ser executado.

Para acessar a camada de adaptação e o repositório de *mashups*, a *engine* de composição comunica-se com o lado do *servidor* do NMMS através de chamadas HTTP. Essas chamadas são assíncronas, ou seja, elas são realizadas e processadas em *background* e não demandam que o *browser* do usuário recarregue a página da interface do NMMS, de acordo com o modelo AJAX. O formato de intercâmbio de dados utilizado nessas chamadas é o JSON.

As requisições ao repositório de *mashups* são *queries* codificadas no padrão JSON-RPC (KOLLHOF, 2005) e são processadas pelo **serviço de persistência** da camada de composição. Ao receber uma chamada, esse serviço identifica a operação desejada, exe-

cuta a *query* SQL apropriada na base de dados do NMMS, codifica a resposta do banco de dados no formato de intercâmbio do NMMS e envia essa resposta para o *browser* do usuário. As operações disponibilizadas por esse serviço incluem:

- Recuperar a lista de *mashups* existentes;
- Recuperar um *mashup*;
- Salvar um *mashup* no repositório;
- Excluir um *mashup* do repositório.

Dado que a *engine* de composição do NMMS foi implementada em JavaScript, a escolha do JSON como formato de intercâmbio de dados tornou-se natural. O motivo disso é que esse formato é um subconjunto da especificação da linguagem JavaScript e, portanto, nativamente suportado pelos interpretadores da mesma. Esse suporte nativo simplifica consideravelmente a manipulação de *wrappers* e *mashups* tanto na engine de composição quanto na camada de apresentação. Além disso, o JSON, como visto no capítulo 2, insere um menor *overhead* nos dados codificados, o que é uma vantagem importante, visto que o lado do cliente do NMMS realiza diversas chamadas assíncronas ao lado do servidor.

A camada de composição foi abordada em detalhes nessa seção, o que permite o entendimento das etapas de planejamento e execução dos *mashups* no NMMS. Como visto no fluxo de atividades da arquitetura, apresentado no capítulo anterior, essa etapas são iniciadas pela camada de apresentação, a qual será abordada com maiores detalhes na seção seguinte.

4.3 NMMS - Camada de Apresentação

Seguindo os princípios da arquitetura apresentada nesse trabalho, a camada de apresentação do NMMS possui dois ambientes distintos, o **Ambiente de Desenvolvimento** e o **Ambiente de Execução**, como pode ser observado na figura 4.7. No ambiente de desenvolvimento, um desenvolvedor pode criar *mashups* utilizando uma interface de composição baseada na manipulação de elementos de interação visuais. Já através do ambiente de execução, um administrador de redes pode invocar *mashups* existentes no NMMS através de uma URL e utilizar esses *mashups* como aplicações Web. Ainda que a composição invocada seja executada pelo NMMS, a participação do sistema na execução é transparente para o administrador.

O ambiente de desenvolvimento do NMMS, que pode ser visto na figura 4.8, foi desenvolvido utilizando HTML e JavaScript segundo o modelo AJAX, através do uso da biblioteca WireIt (ABOUAF, 2007). Na camada de apresentação, essa biblioteca possibilitou o desenvolvimento de uma interface de composição baseada na manipulação e interconexão de elementos visuais denominados **módulos**. O conceito de módulo da camada de apresentação é análogo àquele empregado na camada de composição, isto é, um módulo é um bloco funcional que executa uma única operação *run* sobre N entradas, retornando N saídas que podem ser passadas para outros módulos. Na interface de execução, esses módulos podem ser manipulados através de *drag'n drop*, ou seja, o usuário arrasta os módulos desejados para a **área de trabalho** da *interface* e cria conectores entre esses módulos para definir sua lógica. A área de trabalho, os módulos, conectores e demais elementos visuais da interface de composição estão representados na figura 4.8 e serão descritos em detalhes a seguir.

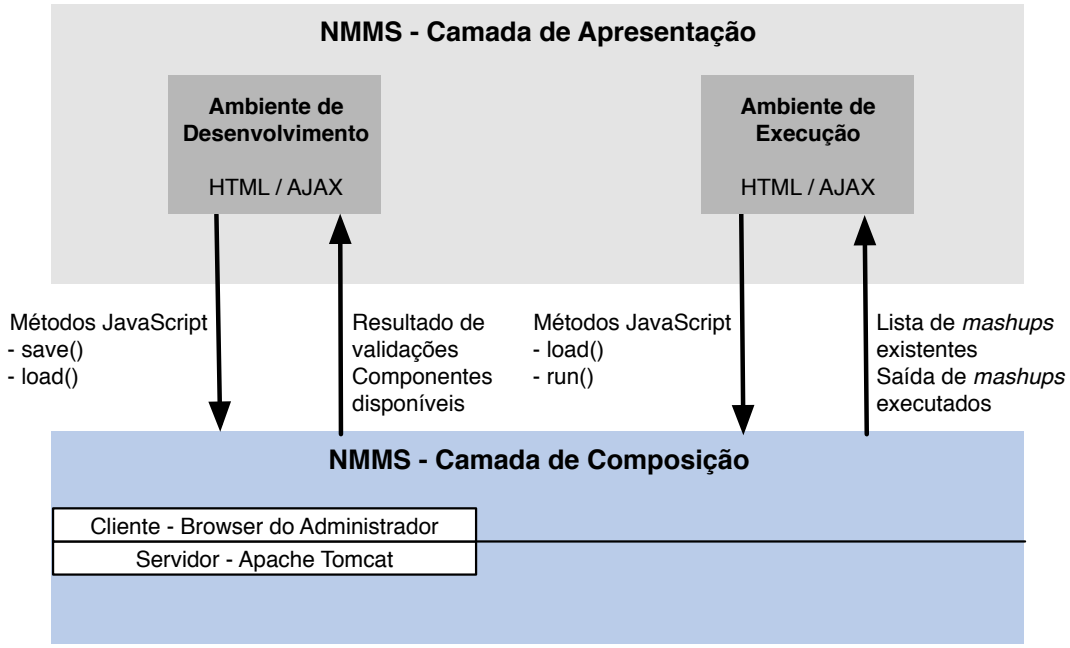

Figura 4.7: A camada de apresentação do *Network Management Mashup System*

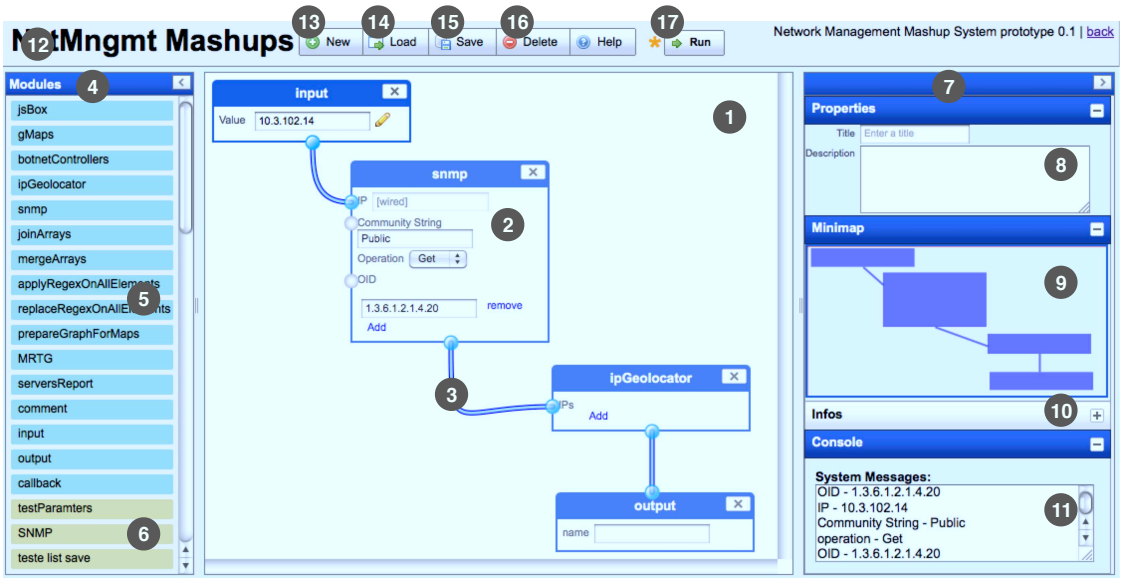

Figura 4.8: Elementos de interação do ambiente de desenvolvimento do NMMS

- **Área de trabalho (1):** Onde o usuário irá manipular e interconectar módulos para definir sua composição;
- **Módulo (2):** Elemento de interação do NMMS que representa um componente de um *mashup*. Os terminais circulares nos lados do bloco visual do módulo representam as entradas e saídas do mesmo;
- **Conexão entre módulos (3)**: Através dessas conexões o *workflow* da composição é descrito na interface. Além disso, os módulos trocam dados através delas. Elas são criadas conectando os terminais dos módulos através de *drag'n drop*;

- **Coluna de módulos existentes (4)**: Essa coluna contém os módulos que o desenvolvedor pode usar para criar seus *mashups*. Os módulos disponíveis são divididos em dois tipos, que são diferenciados por cores:

  Componentes internos (5): Os módulos de cor azulada na coluna de módulos correspondem aos componentes do sistema de *mashups*, como *operadores* e *wrappers*;

  ***Mashups* de gerenciamento (6)**: Os módulos de cor amarela representam os *mashups* existentes no NMMS que podem ser utilizados como componentes em outras composições;

- **Coluna de informações (7)**: A coluna da direita do ambiente de desenvolvimento contém informações sobre a composição sendo desenvolvida:

  Propriedades (8): A área denominada *Properties* guarda as propriedades do *mashup* em desenvolvimento. As propriedades suportadas são nome (*title*) e descrição (*description*);

  Minimapa (9): Mapa da composição atualmente na área de trabalho. Esse mapa serve como um resumo do *mashup* sendo desenvolvido e é navegável, permitindo que o usuário se desloque rapidamente entre regiões de uma composição complexa sem a necessidade de usar barras de rolagem;

  Informações gerais (10): Contém informações gerais sobre o uso do NMMS como, por exemplo, quais são os módulos utilizados para definir as entradas e saídas dos *mashups* criados;

  Console (11): O console é uma área de texto dinâmica através da qual o NMMS exibe *feedback* para o desenvolvedor *mashups*. As informações exibidas no console incluem, por exemplo, o *módulo* atualmente sendo executado pela *engine* e o resultado da validação de uma determinada composição;

- **Barra de operações (12)**: A barra superior da interface contém botões compreendendo as operações que o NMMS realiza sobre composições. Cada botão equivale a uma operação e, quando pressionado, invoca um método JavaScript que realiza a operação, comunicando-se com a camada de composição quando necessário. Os botões de operação são os seguintes:

  ***New* (13)**: Limpa o conteúdo da área de trabalho e inicia a criação de um novo *mashup*;

  ***Load* (14)**: Possibilita a recuperação de *mashups* salvos no NMMS, através da tela vista na figura 4.9. Essa tela contém uma lista das composições existentes no NMMS e um campo de filtragem dessa lista por nome. A composição selecionada na tela de *Load* será aberta na área de trabalho, onde ela pode ser editada e executada;

  ***Save* (15)**: Inicia o *mashup planning* do NMMS. Quando esse botão é pressionado, o ambiente de desenvolvimento solicita que a *engine* de composição valide a composicão e, caso ela seja válida, salve-a no repositório de *mashups*;

  ***Delete* (16)**: Remove o *mashup* atualmente aberto do repositório de *mashups* e limpa a área de trabalho;

  ***Run* (17)**: Solicita que a *engine* de composição a execute o *mashup* atualmente na área de trabalho. Caso esse *mashup* ainda não tenha sido validado, a *engine* de

execução realizará uma validação antes de executá-lo. Ao final dessa execução, a saída da composição executada é impressa no console. Além disso, caso tal saída seja uma página Web, essa página é aberta em uma nova aba/janela do navegador do usuário. A existência desse botão e da operação *run* no ambiente de desenvolvimento permite que a criação de *mashups* siga o modelo de prototipação descrito no capítulo anterior, possibilitando que o desenvolvedor utilize apenas o ambiente de desenvolvimento para realizar mútliplas validações e execuções de composições parciais até que o *mashup* em desenvolvimento atinja o comportamento desejado.

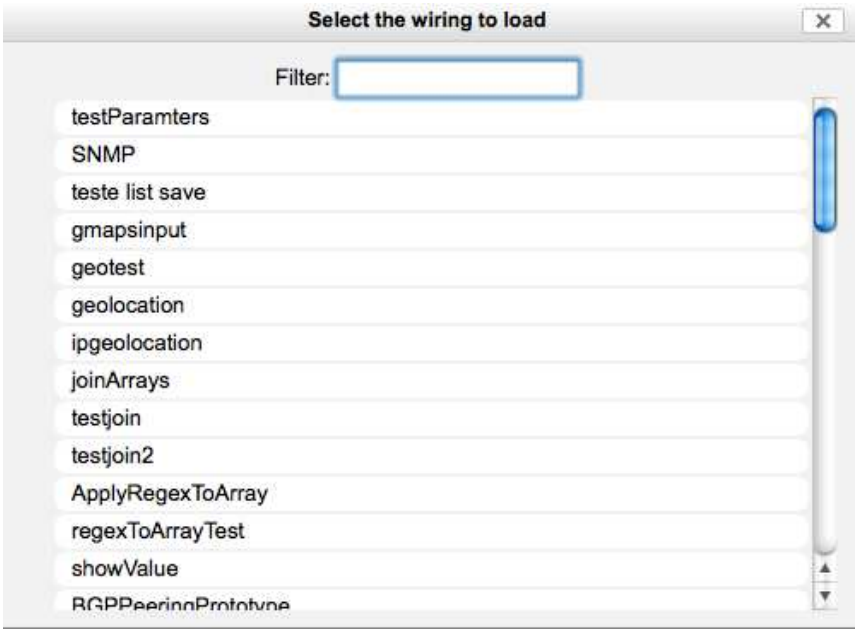

Figura 4.9: Tela de recuperação de *mashups* do NMMS

Quando um *mashup* é salvo no repositório do NMMS, uma URL é criada para o mesmo. Essa *URL* permite que o *mashup* seja acessado através do ambiente de execução. Esse ambiente processa a requisição HTTP gerada pelo acesso à URL de uma composição. Ao receber essa requisição, o ambiente de execução comunica-se com a camada de composição através de métodos JavaScript, que irá recuperar composição solicitada da base de dados e invocar a *engine* do NMMS, a qual, por sua vez, executará o *mashup*. Essa comunicação é transparente para o administrador de redes, que somente enxergará o resultado da execução do *mashup* solicitado. Caso esse *mashup* defina entradas, elas podem ser codificadas como parâmetros da sua URL. Por exemplo, se uma composição denominada *myMashup* define um parâmetro de entrada *input*, sua URL será:

```
http://<Endereço do NMMS>/environment/runtimeEnvironment.html?c=
"myMashup"&"input"="entradaParaMyMashup"
```

Como visto anteriormente, o usuário também pode usar o botão *Run* do ambiente de desenvolvimento para executar o *mashup*. Para tal, ele pode arrastar a composição desejada da barra de *módulos* para a área de trabalho, fazendo com que a interface do NMMS crie o elemento de interação de reuso para a composição. Ao criar o elemento de interação de reuso, a interface do NMMS automaticamente inclui um formulário contendo as

entradas necessárias para executar o *mashup*, como pode ser visto na figura 4.10. Ao pressionar o botão *Run*, o *mashup* é executado recebendo como entrada os dados preenchidos no formulário gerado.

Figura 4.10: Elemento de interação de reuso gerado pelo NMMS

Nessa seção, foi visto como administrador de redes utiliza as interfaces do NMMS, completando a discussão das camadas arquiteturais do *sistema*. Além dessas camadas, também é importante discutir os elementos de interação implementados no NMMS. Eles são vistos em detalhes na próxima seção.

4.4 Elementos de Interação do *Network Management Mashup System*

Como já visto anteriormente, elementos de interação consistem em elementos visuais que abstraem os componentes de um *mashup*. Na camada de apresentação do NMMS, cada módulo é um elemento de interação. Para criar *mashups* de gerenciamento no sistema, o desenvolvedor de *mashups* vai manipular e conectar esses elementos de interação para definir a lógica da sua composição. A seguir, alguns dos principais elementos do NMMS são apresentados e classificados:

- ***Maps***: O *maps* é um elemento de interação tanto de adaptação quanto visual. Ele é um elemento de interação visual pois sua função é possibilitar que o usuário apresente o conteúdo do seu *mashup* em formato de mapa interativo. Entretanto, esse elemento também é considerado de adaptação por abstrair o *wrapper* para serviços de mapas *online*;

- ***Geolocation***: O *geolocation* é um elemento de adaptação que abstrai o *wrapper* de serviços de geolocalização do NMMS. Esse elemento possibilita que o *mashup* descubra informações geográficas sobre determinado endereço IP;

- **SNMP**: O SNMP é um elemento de adaptação que abstrai o *wrapper* para o protocolo SNMP. Tal elemento permite a criação de *mashups* capazes de se comunicar via SNMP com elementos gerenciados como roteadores e *switches*;

- **JsBox**: Elemento de *scripting* que permite que o desenvolvedor utilize *scripts* desenvolvidos na linguagem JavaScript nos seus *mashups*. Esse elemento permite que que administradores de redes que também possuem habilidade em desenvolvimento de *software* façam uso dessa habilidade na criação de *mashups* de gerenciamento;

- **RegExp**: O RegExp é um elemento de interação que possibilita a aplicação de expressões regulares sobre os dados do *mashup*. Como esse elemento abstrai operações sobre dados, ele é considerado um elemento de operação. A partir dele,

foi possível criar pequenos *mashups* auxiliares que utilizam expressões regulares específicas para extrair, por exemplo, endereços IP ou ASNs de um determinado conjunto de dados. Esses *mashups*-operadores derivados do RegExp são de mais simples uso por um usuário leigo em desenvolvimento. Eles podem ser usados em outras composições através de elementos de reuso.

É importante ressaltar que esses são apenas exemplos de elementos de interação disponíveis no NMMS e não correspondem ao conjunto total de elementos do sistema. Esse conjunto é evolutivo, visto que novos elementos de interação podem surgir dependendo das necessidades dos administradores utilizando o sistema. Muitos dos elementos apresentados surgiram dessa forma, a partir de necessidades encontradas nos estudos de caso abordados na próxima seção, que trata dos resultados obtidos com o uso do NMMS em cenários reais de gerenciamento de redes.

5 AVALIAÇÃO

Para avaliar tanto a aplicação de *mashups* no gerenciamento quanto o protótipo *Network Management Mashup System* apresentado no capítulo anterior, o presente capítulo discute estudos de caso onde necessidades reais de gerenciamento foram atendidas por *mashups* criados no NMMS. O impacto da utilização do NMMS na criação desses *mashups* é avaliado através da comparação com o processo de desenvolvimento *ad hoc* (*i.e.*, sem o auxílio de sistemas de *mashups*) das mesmas composições. Já os *mashups* de gerenciamento em si são avaliados através de uma análise qualitativa baseada nos seguintes critérios:

- Esforço de implementação;
- Esforço de utilização;
- Confiabilidade;
- Extensibilidade;
- Flexibilidade.

Os estudos de caso, que serão abordados em detalhes nas próximas seções, são os seguintes:

- ***Peering* BGP**: Consiste na utilização de um *mashup* para a visualização integrada de informações envolvidas no gerenciamento dos domínios administrativos autônomos (*Autonomou System* - AS) que compõem a Internet. A conexão entre esses domínios é controlada pelo protocolo BGP (*Border Gateway Protocol*) (REKHTER; LI, 1995) e dois ASes conectados através desse protocolo são denominados *Peers* (BATTISTA; REFICE; RIMONDINI, 2006);

- ***Detecção de Botnets***: Trata da detecção de *botnets*, redes massivas de computadores conectados à Internet e infectados por *malwares*, as quais são utilizadas para propósitos criminosos como *spamming*, *phishing* e ataques de negação de serviço distribuídos (*Distributed Denial of Service* - DDoS) (PAUL BACHER THORSTEN HOLZ; WICHERSKI, 2005). O *mashup* desse estudo de caso fornece uma visualização integrada de informações a respeito dos controladores dessas redes. Tais informações são obtidas através da correlação de diferentes ferramentas de detecção distintas.

5.1 Estudo de Caso 1 - *Peering BGP*

O primeiro estudo de caso, referenciado nessa dissertação como *Peering* BGP, diz respeito à comunicação entre os diferentes domínios administrativos independentes que compõem a Internet. Esses domínios, também denominados sistemas autônomos (ASes), pertecem em sua maioria a grandes provedores de serviços (*Internet Service Provider* - ISP) como, por exemplo, a Rede Nacional de Pesquisa (RNP). A denominação *sistemas autônomos* advém do fato de um administrador de um determinado AS poder definir políticas de gerenciamento (*e.g.*, políticas de roteamento, de qualidade de serviço, de prioridades de tráfego) independentemente dos demais ASes da Internet (HAWKINSON; BATES, 1996).

A conexão entre dois ASes é realizada quando um deles anuncia rotas para o outro através do protocolo BGP (*Border Gateway Protocol*). Essas rotas consistem em conjuntos de prefixos IP nos quais o AS anunciante está preparado para receber tráfego. Dois ASes conectados dessa forma são denominados *peers* e a conexão entre esses domínios administrativos é denominada *Peering* BGP (BATTISTA; REFICE; RIMONDINI, 2006).

Normalmente, o comportamento esperado do relacionamento de dois ASes é governado por Acordos de Nível de Serviço (*Service Level Agreement* - SLA) entre as entidades que controlam esses domínios. Alguns dos principais cenários problemáticos enfrentados por administradores de ASes ocorrem quando esse relacionamento inter-domínio se comporta de maneira inesperada, ou seja, quando SLAs não estão sendo cumpridos (BEZERRA et al., 2010). Um exemplo desse tipo de cenário ocorre quando um *peer* BGP anuncia um número de rotas diferente do acordado, visto que esse anúncio pode interferir com tabelas de roteamento do AS que recebe as rotas. Outro caso problemático ocorre quando um AS envia tráfego muito acima do esperado para seus *peers*, o que pode causar congestionamentos de larga escala.

Ambos os cenários problemáticos exemplificados acima podem acontecer devido a elementos de rede (*e.g.*, roteadores BGP) mal configurados e/ou com problemas de funcionamento. Independente da causa, é importante que esses problemas de *peering* BGP sejam resolvidos de forma ágil, dado o grande volume de tráfego trocado entre ASes e os altos custos financeiros associados a essas trocas. Além disso, como os ASes são subredes da própria Internet, é provável que problemas na comunicação entre esses domínios afete uma enorme quantidade de usuários.

Para administrar adequadamente um sistema autônomo e detectar problemas no domínio, um administrador de redes deve lidar com uma quantidade elevada de informações obtidas a partir de ferramentas de gerenciamento distintas. Dentre essas informações, as mais importantes são as seguintes:

- **Informações administrativas sobre os *peers* BGP**: Consistem no conjunto de *peers* do AS gerenciado e informações relevantes sobre os mesmos como, por exemplo, seus identificadore únicos (*Autonomous System Number* - ASN) e os endereços IP dos seus roteadores BGP. Essas informações fazem parte do módulo da MIB denominado BGP4-MIB (WILLIS; BURRUSS; CHU, 1994) e podem ser obtidas via SNMP;

- **Número de rotas anunciadas**: Inclui tanto o número de rotas anunciados pelo AS gerenciado para outros domínios quanto as rotas anunciadas por cada *peer*. Como essa informação não é fornecida diretamente pelo módulo BGP4-MIB (DOS SANTOS et al., 2010), é necessário calcular esse número através do processamento da

tabela de roteamento BGP. Entretanto tal processamento é complexo e custoso, visto essa tabela é global (*i.e.,* compreende todas as conexões entre os ASes que compõem a Internet). Uma forma alternativa de obter essa informação é utilizar extensões do módulo BGP4-MIB como a proposta em (CERON et al., 2009). Essa alternativa requer, no entanto, que o agente SNMP do roteador gerenciado suporte as extensões do BGP4-MIB;

- **Tráfego trocado**: Corresponde ao tráfego trocado entre o AS gerenciado e seus *peers* BGP. Há algum suporte para a obtenção desse tipo de informação via SNMP, através dos objetos da MIB `IfInOctets` e `IfOutOctets`, definidos pelo módulo MIB-II (MCCLOGHRIE; ROSE, 1991). Para a utilização desses objetos é necessário descobrir, através da correlação entre objetos dos módulos MIB-2 e BGP4-MIB, quais interfaces de rede do roteador gerenciado conectam-se aos *peers* BGP. Além disso, é importante considerar que os objetos de tráfego do módulo MIB-II mencionados correspondem ao número total de octetos (*i.e., bytes*) trocados no momento da requisição SNMP, ou seja, esses objetos fornecem informações isentas de análise temporal. Para obter uma informação de tráfego que caracterize adequadamente o comportamento da rede em um determinado intervalo de tempo, é necessário o uso de ferramentas como o MRTG (OETIKER, 2008);

- **Localização geográfica dos *peers***: Consistem em informações como, por exemplo, latitude, longitude, cidade e país onde estão localizados os roteadores BGP dos *peers* do AS gerenciado. Esse tipo de informação permite uma compreensão geográfica do *peering* BGP, a qual pode ser útil para, por exemplo, avaliar a latência da conexão com diferentes *peers*. Essas informações podem ser obtidas diretamente com as entidades que administram os *peers* ou através de serviços de geolocalização de endereços IP.

Apesar da integração das informações e ferramentas descritas acima ser uma necessidade do administrador de redes, até a redação do presente trabalho não foi encontrada nenhuma ferramenta específica para tal fim. A subseção seguinte descreverá como a aplicação de *mashups* atende a essa necessidade, descrevendo a criação de um *mashup* de gerenciamento de *peering* BGP.

5.1.1 *Mashup* Desenvolvido

A presente subseção descreve um *mashup* criado a partir da composição das informações e ferramentas relativas ao cenário de *peering* BGP. Inicialmente, o processo de desenvolvimento desse *mashup* utilizando o NMMS será abordado. Em seguida, o resultado da composição será apresentado.

A figura 5.1 exibe a modelagem do *workflow* seguido pelo *mashup* de *peering* BGP, descrita na linguagem *Business Process Modeling Notation* (BPMN) (SILVER, 2009). Esse *mashup* agrega informações sobre a conexão de roteadores BGP locais (*i.e.*, pertencentes ao AS gerenciado) com seus *peers* remotos (*i.e.,* outros ASes) em uma página Web baseada em mapas. Os módulos do NMMS utilizados nesse *workflow* são os seguintes:

- **SNMP**: Módulo responsável pela recuperação de informações de dispositivos gerenciados através do protocolo SNMP. Recebe como entrada o endereço do dispositivo alvo, a *string* de comunidade SNMP de acesso a esse dispositivo, a operação a ser realizada e os OIDs (*Object Identifier*) dos objetos da MIB que devem recuperados. Retorna os valores dos objetos solicitados;

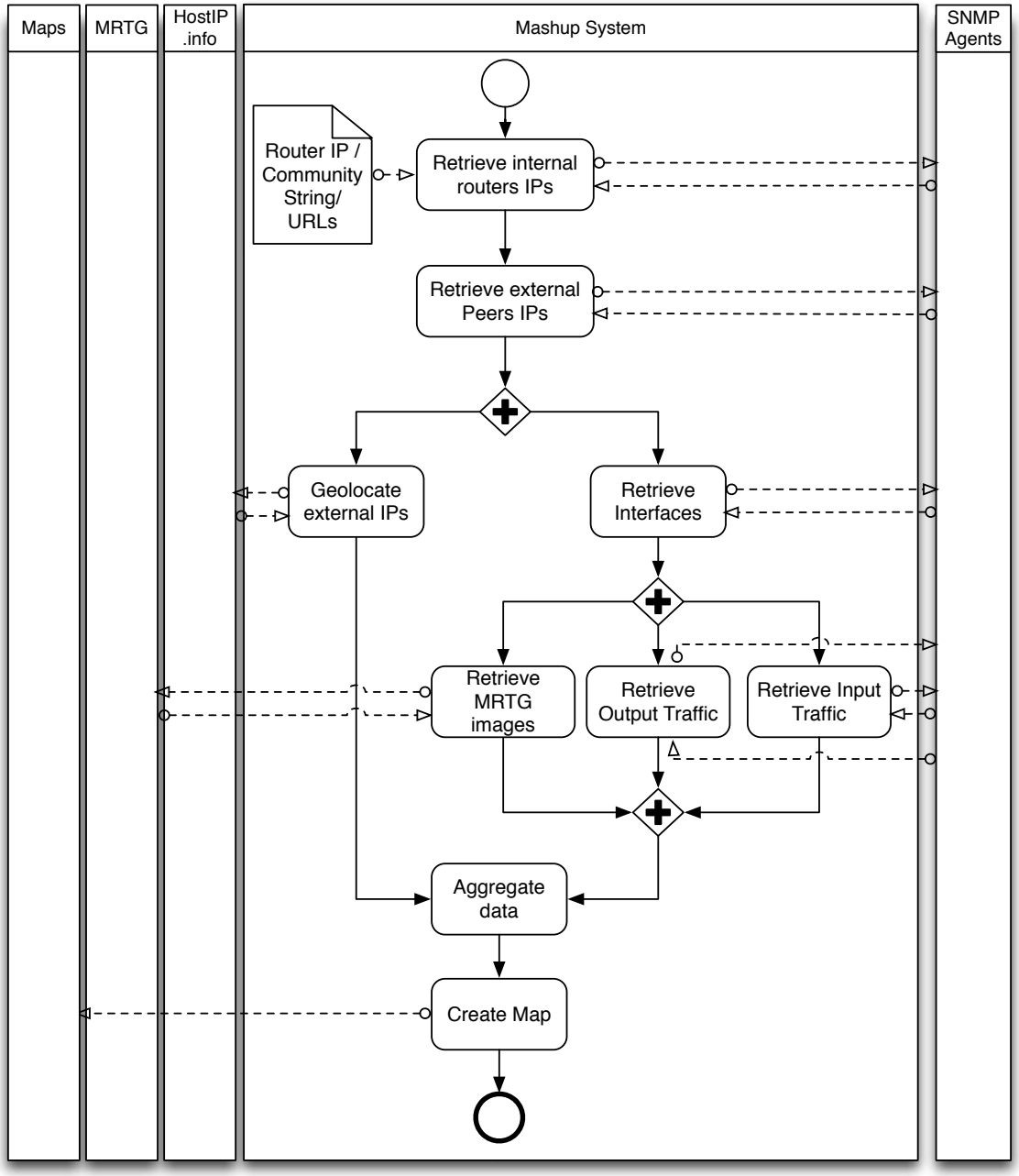

Figura 5.1: *Workflow* do *mashup Peering* BGP *Map*

- ***Geolocation***: Módulo de adaptação para o serviço de geolocalização *hostip.info*. Recebe como entrada um endereço IP e retorna as coordenadas geográficas aproximadas (*i.e.*, latitude e longitude) desse endereço;

- ***Image Extraction***: Módulo responsável por extrair imagens de *front ends* Web disponibilizados por ferramentas de gerenciamento. No *mashup* de *peering* BGP, esse módulo é utilizado para extrair imagens de uma instância da ferramenta MRTG. A entrada para esse módulo é a URL do *front end* do MRTG, que pode estar ser tanto um endereço local quanto externo (*i.e.*, da Internet). A saída desse módulo é um o documento HTML contendo as imagens solicitadas;

- ***Graph***: Módulo de reuso utilizado para a agregação das informações coletadas em

um grafo que representa as conexões entre um roteador BGP e seus *peers*. Esse operador recebe como entrada informações geográficas e informações obtidas via SNMP, principalmente das MIBs BGP-4 e MIB2. Com elas, ele compõe um grafo onde o nó central é o roteador BGP e os demais nós são *peers*. A saída desse módulo é uma *string* em formato JSON representando o grafo. Esse módulo também pode receber figuras obtidas pelo módulo *Image Extraction* e associá-las a *peers*;

- ***Maps***: Esse é um módulo de adaptação e visualização responsável por criar um mapa visual de rede e anotá-lo informações administrativas. Para tal, o módulo recebe como entrada uma descrição da rede no formato JSON. O retorno desse módulo é uma página HTML/JavaScript que exibe o mapa gerado. Para gerar o mapa, o módulo *Maps* implementado no NMMS utiliza a API do Google Maps (GOOGLE, 2005);

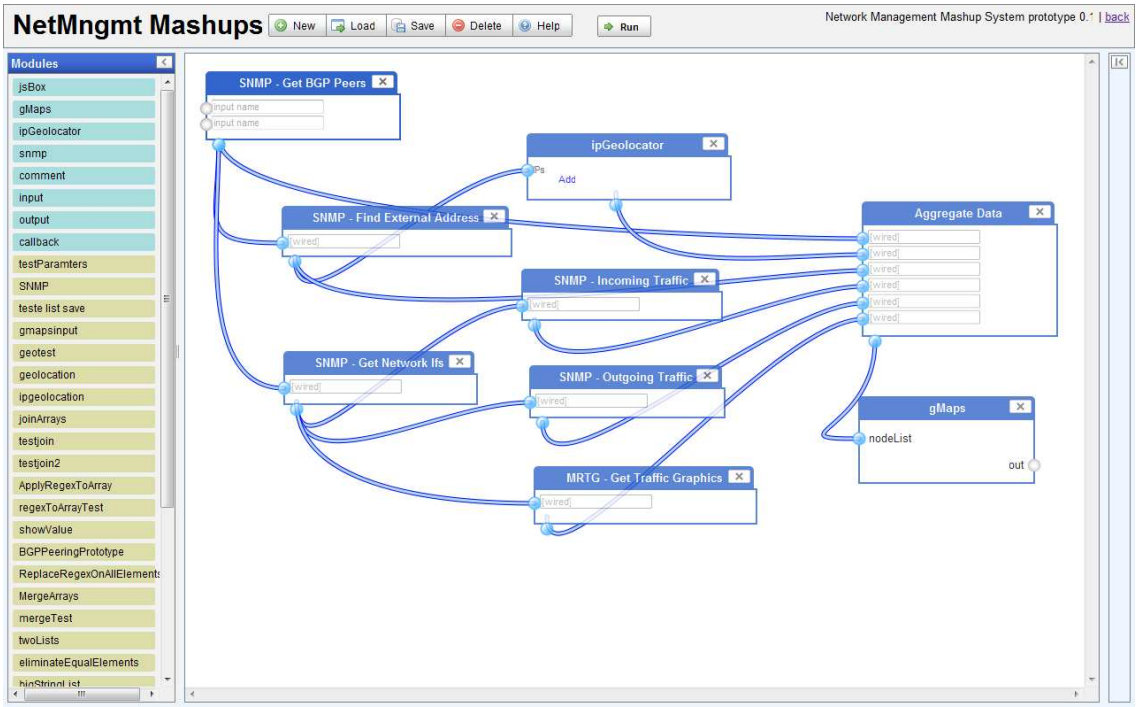

Figura 5.2: *Mashup* de *peering* BGP desenvolvido na interface de composição do NMMS

A figura 5.2 mostra o *mashup* desenvolvido na interface de composição do NMMS. Ela mostra o equivalente, no sistema, ao *workflow* descrito pela figura 5.1. Esse *workflow* recebe os seguintes parâmetros de entrada:

- Endereço IP do roteador BGP local;
- *String* de comunidade SNMP de acesso ao roteador local;
- URL da instância da ferramenta *MRTG* que colhe tráfego no roteador BGP.

A partir dos parâmetros listados, o *workflow* do *mashup* é iniciado pelo módulo *SNMP*, que recupera o *peers* BGP do roteador local obtendo, especificamente, os endereços IP internos e os ASNs desses *peers* na tabela de roteamento desse roteador. A partir dos endereços internos, o módulo SNMP recupera quais os endereços IP externos (*i.e.*, endereços válidos na Internet) dos *peers* e em quais interfaces de rede do roteador local

esses *peers* estão conectados. Os endereços externos dos *peers* são, então, submetidos ao módulo *Geolocation*, que vai determinar as informações geográficas dos mesmos. As interfaces, por sua vez, são submetidas juntamente com os endereços dos *peers* para o módulo *Image Extraction*, o qual obtém do MRTG gráficos com estatísticas de anúncio de rotas e tráfego trocado entre o roteador local e cada *peer* BGP. O conjunto de informações obtido (*peers*, informações geográficas, tráfego e rotas anunciadas) é, então, submetido ao módulo *Graph*. Tal módulo faz um *merge* dessas informações, criando o grafo da rede de *peering* BGP a partir da correlação das mesmas. Finalmente, esse grafo é submetido ao módulo *Maps*, que cria a visualização, marcando pontos em um *mapa mundi* representando tanto o AS local quanto seus *peers*, fazendo as conexões entre esses pontos e, finalmente, anotando cada ponto e cada conexão do grafo com suas respectivas informações de tráfego e anúncio de rotas. Essa visualização pode ser vista na figura 5.3.

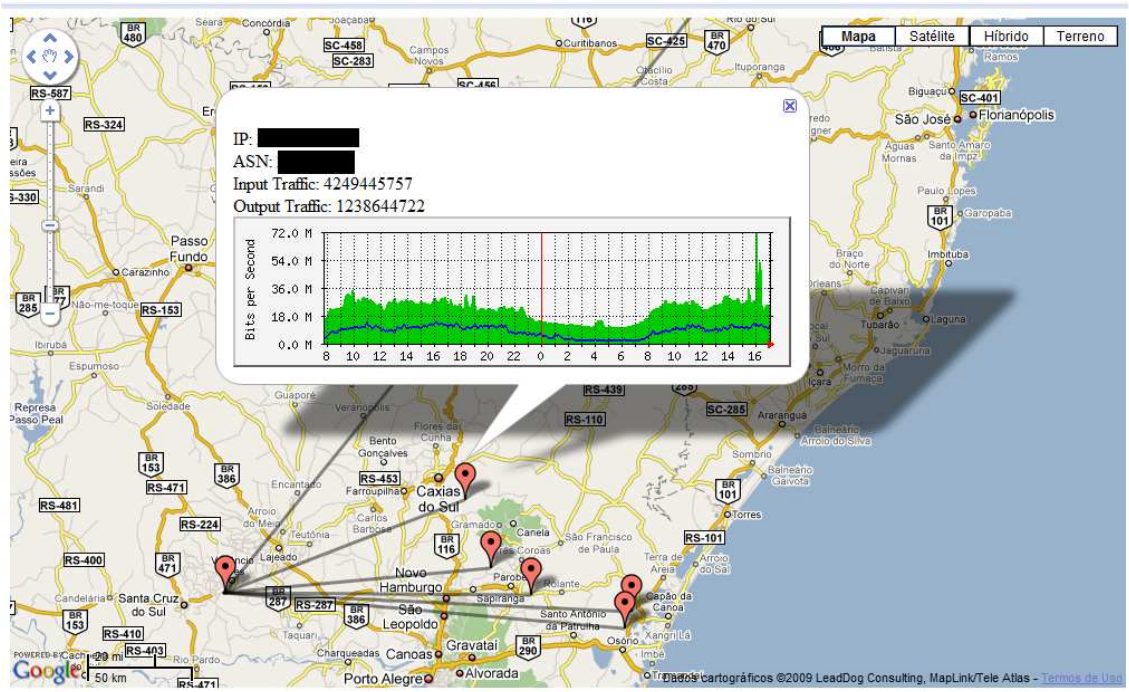

Figura 5.3: Ambiente de execução do NMMS exibindo o resultado do *mashup* de *peering* BGP

No ambiente de execução do *mashup* criado, visto na figura 5.3, pode-se ver o resultado da composição das informações de *peering* BGP em formato de mapa. Os marcadores do mapa correspondem aos *peers* BGP coletados, sendo que o AS gerenciado corresponde ao marcador central e os demais marcadores, aos ASes remotos. O administrador de *redes* pode interagir com esse mapa, ajustando sua posição e nível de *zoom* para, por exemplo, visualizar apenas um determinado estado ou país. Interações com elementos que representam a rede visualizada (*e.g.*, um marcador que representa um AS), realizadas através de cliques de *mouse*, permitem que esse administrador enxergue informações detalhadas sobre uma conexão BGP, tais como AS conectado, tráfego e rotas anunciadas.

O *mashup* criado oferece mais possibilidades além das interações vistas acima. Por exemplo, é possível inserir múltiplos roteadores *BGP* locais no mapa para a visualização de cenários de *peering* BGP mais complexos. Também é possível salvar o *mashup* e recarregá-lo através do NMMS. A página do *mapa* em si não é salva, ela é criada dinamicamente a partir da descrição da composição salva no NMMS. Quando uma instância

específica do *mashup* de *peering* BGP é recarregada, a composição é re-executada pelo NMMS com os parâmetros salvos, recriando o mapa. Finalmente, existe a possibilidade de programar o *mashup* para se atualizar-se automaticamente em tempo de execução.

Nessa subseção, foi discutido como aplicar a tecnologia de *mashups* ao estudo de caso *peering* BGP, através da criação de um *mashup* de gerenciamento integrando as informações e ferramentas envolvidas nesse *peering* e utilizando, para essa criação, o protótipo NMMS. A próxima subseção, por sua vez, discutirá os resultados obtidos com o uso do NMMS na solução desse estudo de caso. Nessa discussão, o processo de desenvolvimento da composição de recursos criada via NMMS será comparado com o desenvolvimento do mesmo *mashup* de forma *ad hoc*. Após essa comparação, o impacto do uso do *mashup* de *peering* BGP pelo administrador de um sistema autônomo será analisado.

5.1.2 Discussão dos Resultados

Para fins de comparação, o *mashup* de *peering* BGP também foi desenvolvido de forma *ad hoc*. Para desenvolver o *mashup* dessa forma, mostrou-se necessário o conhecimento de conceitos fundamentais de lógica de programação tais como iterações, laços, lógica *booleana* e operações sobre grafos. Além desses conceitos, o processo de desenvolvimento *ad-hoc* demandou o domínio das seguintes tecnologias:

- **Linguagens de programação**: Java e JavaScript;
- **Formato de dados**: JSON;
- **Protocolos**: HTTP e SNMP;
- **APIs *Online***: APIs de geolocalização (*i.e.*, API RESTful do serviço *hostip.info*) e API do *Google Maps* (GOOGLE, 2005).

Dentro do contexto de cada uma dessas tecnologias, diversos conceitos de caráter técnico foram empregados como, por exemplo, o uso de *sockets* (WINETT, 1971) para a comunicação através de protocolos de rede. Já na criação do *mashup* utilizando o NMMS, somente foram necessários conhecimentos relacionados ao gerenciamento de redes, como o funcionamento básico do SNMP e o cenário de *peering* BGP a ser abordado pela composição. Os *wrappers* do sistema encarregaram-se de isolar e abstrair as tecnologias e aspectos técnicos envolvidos na composição de recursos. Pode-se perceber, portanto, que a utilização do NMMS simplificou a criação do *mashup* do estudo de caso, efetivamente mitigando a necessidade de dominar tecnologias e conhecimentos relacionados ao desenvolvimento de *software*.

Além de mitigar a necessidade de dominar diversas tecnologias, a utilização do NMMS também propiciou uma redução do esforço de criação do *mashup* de *peering* BGP em relação à abordagem de desenvolvimento *ad hoc*. Essa redução pode ser observada ao comparar a manipulação de recursos externos em ambas as abordagens. Por exemplo, para o caso da manipulação de um recurso de gerenciamento de redes, pode-se observar a obtenção dos *peers* BGP via SNMP nas duas implementações. A figura 5.4 mostra o trecho de código Java utilizado na implementação *ad hoc* para recuperar esses *peers*. Na implementação via NMMS, as operações realizadas pelo código mostrado foram abstraídas pelo elemento de interação SNMP, que pode ser visto no trecho demarcado pela região **1** da figura 5.5. Comparando o trecho de código com a composição no NMMS, é possível observar que o sistema de *mashups* não só reduziu o número de operações necessárias como substituiu um código de sintaxe complexa por uma linguagem visual baseada

em blocos, mais adequada a um administrador de redes leigo em desenvolvimento de *software*.

```
1        BGPRouter r = new BGPRouter(targetAddress);
2        LinkedList<Peer> tempPeers = new LinkedList<Peer>();
3
4        double[] coordinates = Geolocator.getCoordinatesFromIP(r.getIp());
5        r.setLat(coordinates[0]);
6        r.setLon(coordinates[1]);
7
8        r.setAsn(SNMPUtil.get("1.3.6.1.2.1.15.2.0", targetAddress, community));
9
10       for (String s : (SNMPUtil.walk("1.3.6.1.2.1.15.3.1.9", targetAddress,
11               community))) {
12           s = s.replace("1.3.6.1.2.1.15.3.1.9.", "");
13
14           String[] response = s.split(" = ");
15           Peer peer = new Peer();
16
17           peer.setIp(response[0]);
18           peer.setAsn(response[1]);
19           double[] coords = Geolocator.getCoordinatesFromIP(peer.getIp());
20           peer.setLat(coords[0]);
21           peer.setLon(coords[1]);
22
23           boolean routerAlreadyListed = false;
24           for (Peer p : tempPeers) {
25               if (p.isSameRouter(peer)) {
26                   p.setIp(p.getIp() + ";" + peer.getIp());
27                   routerAlreadyListed = true;
28                   break;
29               }
30           }
31
32           if (!routerAlreadyListed) {
33               tempPeers.add(peer);
34           }
35
36       }
```

Figura 5.4: Código Java do *mashup ad hoc* para recuperar os *peers* BGP

Para comparar a utilização de um recurso Web nas diferentes abordagens de implementação do *mashup* de *peering* BGP, pode-se observar o caso da criação do mapa de interconexão dos ASes (*i.e.*, a visualização do *mashup*). O código JavaScript mostrado na figura 5.6 mostra a criação desse mapa na implementação *ad hoc* do *mashup*. Na implementação via NMMS, por sua vez, a criação do mapa é representada pela região **2** da figura 5.5. Comparando as implementações, é possível perceber que o NMMS também reduziu o esforço necessário nesse caso, diminuindo o número de operações definidas pelo desenvolvedor de *mashups* e oferecendo uma interface mais amigável para a utilização do recurso externo.

Outra vantagem da implementação via NMMS sobre a abordagem *ad hoc* é a reusabilidade da composição criada. A implementação do *mashup* de *peering* BGP via NMMS segue o padrão de interconexão de módulos adotado pelo sistema, sendo naturalmente compreensível para um administrador que utiliza a ferramenta. Caso esse mesmo administrador queira reutilizar ou modificar a implementação *ad hoc*, ele precisará consultar documentação para compreender tanto o raciocínio quanto o código do autor original do *mashup*. Entretanto, não há qualquer garantia de que o autor de um *mashup ad hoc* documentará seu código. Em um cenário onde *mashups* são criados rapidamente para atender necessidades situacionais urgentes, é pouco provável que tempo e esforço sejam dedicados à documentação do código criado.

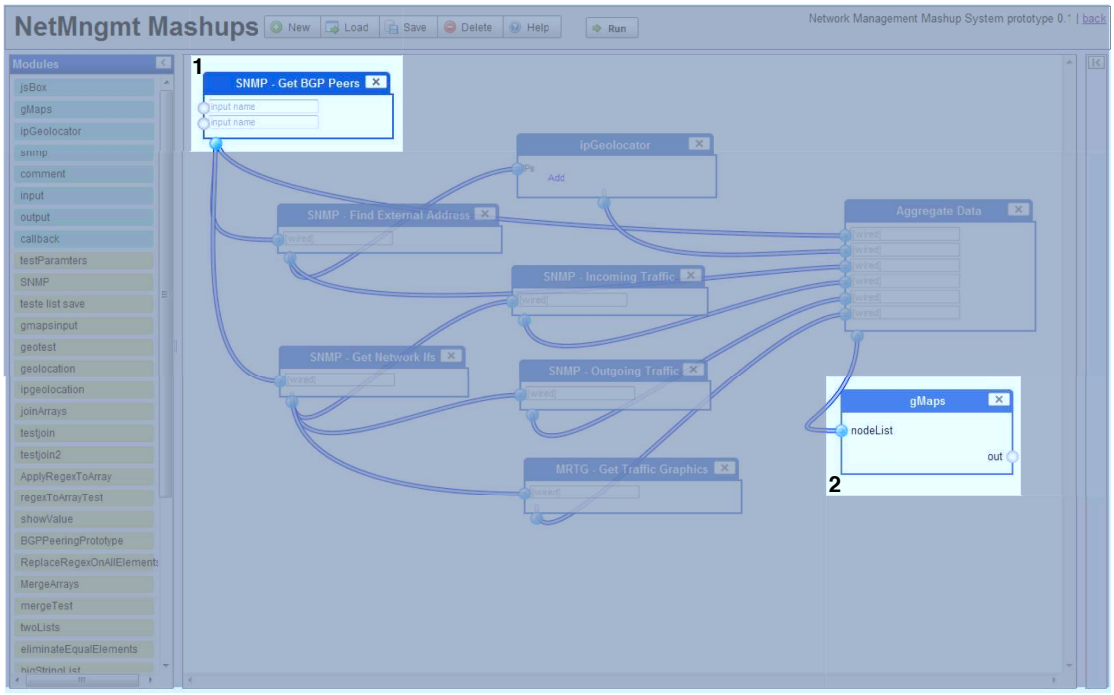

Figura 5.5: Implementação do *mashup* no NMMS: **1-** Obtenção dos *peers* BGP; **2-** Criação do mapa.

Analisando o *mashup* criado em si, o principal benefício exibido pelo mesmo está relacionado a interface baseada em *mapa*. Ela exibe de forma direta uma representação de toda a rede de ASes conectados a um determinado roteador BGP, fornecendo ao administrador uma visão geral do cenário de *peering* BGP. O *mashup* vai além dessa visão geral, permitindo que esse administrador obtenha, através de poucos cliques, informações de tráfego e anúncios de rotas sobre cada um dos seus *peers*. Tradicionalmente, para obter essas informações sobre *peering* BGP, o administrador teria que realizar diversos passos de interação (*e.g.*, preenchimentos de campos de dados, cliques) em várias interfaces de ferramentas distintas, correlacionar essas informações através de *scripts* complexos e, finalmente, criar uma visualização adequada para o resultado dessa correlação utilizando, por exemplo, *softwares* de criação de imagens (DOS SANTOS et al., 2010). O NMMS reduz o número de passos de interação necessários ao possibilitar que esse processo de coleta, correlação e visualização de informações seja realizado através de duas interfaces que não requerem habilidades técnicas (*i.e.*, a interface de composição do NMMS e a interface visual do *mashup*). Além disso, a popularidade de *mashups* baseados em mapa (ZANG; ROSSON; NASSER, 2008) e de mapas de rede como ferramenta de gerenciamento (LEINWAND; CONDROY, 1996) são fortes indícios da alta usabilidade desse tipo de interface visual, usabilidade essa que é herdada pelo *mashup* criado.

Além dos benefícios de usabilidade discutidos, o *mashup* de *peering* BGP também mostrou-se flexível o suficiente para ser estendido para outros cenários. A subseção seguinte discute como uma extensão desse *mashup* foi utilizada para resolver uma generalização do estudo de caso apresentado.

```
function loadMap(){
        var router, comm;
        router = document.getElementById("loadField").value;
        writeOnConsole("Load: " + router);
        var peers = getPeerArrayData(router, null, true);
        var bgpCoord;
        var markers = new Array();
        var markersCount = 0;
        var lines = new Array();
        var linesCount = 0;
        var x;
        for (x in peers) {
            var marker = new GMarker(new GLatLng(peers[x].lat, peers[x].lon));
            marker.enableDragging();
            addInfoToMarker(marker, peers[x]);
            if (peers[x].isLocalBgpRouter == true) {
                bgpCoord = peers[x];
            }
            else {
                var vertices = new Array();
                vertices[0] = new GLatLng(bgpCoord.lat, bgpCoord.lon);
                vertices[1] = new GLatLng(peers[x].lat, peers[x].lon);
                var midpoint = getMidPoint(vertices[0], vertices[1]);
                var lines = new Array();
                var black = "#000000";
                if (peers[x].incoming >= 0) {
                    var incoming = new Array();
                    var red = "#FF0000";
                    incoming[0] = vertices[1];
                    incoming[1] = new GLatLng(midpoint[0], midpoint[1]);
                    var lineThickness = 10;
                    var color = black;
                    var inLine = new GPolyline(incoming, color, lineThickness);
                           map.addOverlay(inLine);
                           var info = "To: "+ bgpCoord.ip +"<br/>Bytes: " + peers[x].
                               incoming;
                           addInfoWindowEvent(map, inLine, info);
                      }
                     if (peers[x].outgoing >= 0) {
                           var outgoing = new Array();
                           outgoing[0] = vertices[0];
                           outgoing[1] = new GLatLng(midpoint[0], midpoint[1]);
                           var outLine = new GPolyline(outgoing, color, lineThickness);
                           var info = "To: " + ("-" + (peers[x].ip.replace(/;/g, "<br> 
                               -"))) + "<br/>Bytes: " + peers[x].outgoing;
                           addInfoWindowEvent(map, outLine, info);
                           map.addOverlay(outLine);
                     }
                   }
                   map.addOverlay(marker);
                }
}
```

Figura 5.6: Código JavaScript do *mashup ad hoc* para criar o mapa, inserir os ASes e as conexões entre eles

5.1.3 Expansão do Estudo de Caso

O estudo de caso de *peering* BGP pode ser generalizado para levar em consideração o cenário onde o *backbone* do sistema autônomo gerenciado é composto por diversos Pontos de Troca de Tráfego (PTT). Esses PTTs atendem a regiões geográficas diferentes, podem ser gerenciados de forma independente dos demais pontos de troca e conectam-se com diferentes *peers* BGP. Como pode ser visto na figura 5.7, nesse cenário existem diferentes usuários interessados em acessar informações sobre o *peering* BGP, os quais têm diferentes privilégios de acesso a tais informações. Esses usuários sáo:

- **Chefe de operações (CO)**: Administrador do AS, responsável por tomar decisões

sobre a capacidade da rede, tais como onde e como melhorá-la;

- **Administrador de PTT**: Responsável por administrar um determinado PTT controlando, por exemplo, o tráfego entre esse ponto e seus *peers* BGP;
- **Cliente**: Representam os *peers* BGP conectados ao AS gerenciado, que trocam tráfego com o mesmo através de um ou mais PTTs;
- **Usuário Final**: Quaisquer outros usuários interessados em informações sobre o AS gerenciado. Esses usuários não possuem privilégios de acesso às informações da rede.

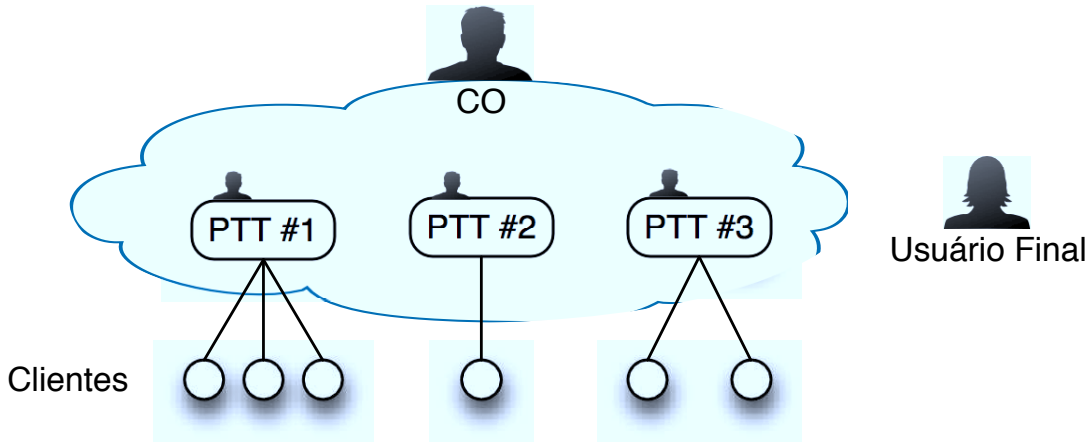

Figura 5.7: Esquema de interesse em informações de *peering* BGP de um AS com múltiplos PTTs

O *mashup* criado para abordar o cenário descrito acima é denominado *Network Traffic Monitoring*. Ele tem como objetivo mostrar informações adequadas para os diferentes usuários descritos, levando em consideração os privilégios de acesso dos mesmos a essas informações. Para tal, o *mashup* divide as informações de tráfego inter-AS nas seguintes categorias:

- **Tráfego detalhado de cada PTT ($\mathbf{PTT_D}$)**: Corresponde ao tráfego trocado com cada cliente por cada um dos PTTs;
- **Tráfego agregado de cada PTT ($\mathbf{PTT_A}$)**: Corresponde ao tráfego agregado que um PTT troca com todos os seus clientes;
- **Tráfego detalhado do AS ($\mathbf{AS_D}$)**: Corresponde a todo o tráfego trocado entre o AS e um determinado cliente. Abrange, portanto, o tráfego trocado entre esse cliente e todos os PTTs;
- **Tráfego agregado do AS ($\mathbf{AS_A}$)**: Corresponde a todo o tráfego gerado e recebido pelo AS gerenciado, ou seja, o tráfego agregado de todos os PTTs com todos os clientes.

A tabela 5.1 mostra os níveis de acesso à informação de tráfego do AS de cada um dos usuários atendidos pelo *mashup* criado. Esses níveis de acesso são explicados a seguir:

Tabela 5.1: Níveis de Acesso a Informação

	AS_A	AS_D	PTT_A	PTT_D
Usuários Finais	X			
Clientes	X	X		
Administradores de PTT	X	X	X	
Chefe de Operações	X	X	X	X

- **Chefe de Operação**: O CO tem acesso a todas as informações de tráfego do AS, tanto detalhadas quanto sumarizadas (*i.e.,* agregadas). Essas informações possibilitam que o chefe de operações tenha uma visão geral do *backbone* gerenciado, permitindo, por exemplo, que o mesmo compreenda padrões de utilização da rede. Esse tipo de visão é importante para planejar novos investimentos na rede e possibilitam que sejam alocados mais recursos para PTTs sobrecarregados ou para atender clientes mais exigentes;

- **Administrador de um PTT**: Os administradores dos PTTs não devem ter acesso às informações de tráfego detalhadas dos demais PTTs, respeitando o princípio do privilégio mínimo (*Principle of Least Privilege*), que é central a boas práticas de segurança da informação. Ao aplicar tal princípio, restringindo o acesso de cada administrador às informações detalhadas do seu PTT, minimiza-se, por exemplo, o risco de vazamento de informações sensíveis sobre a rede;

- **Clientes**: Como cada cliente pode conectar-se a múltiplos PTTs distintos dentro do AS gerenciado, é importante que esse cliente tenha acesso às informações de tráfego que ele troca com cada um desses PTTs. Um cliente não deve ser capaz de acessar informações relativas a outros clientes, sejam elas agregadas ou detalhadas;

- **Usuários Finais**: Devem ser capazes apenas de enxergar informações generalizadas sobre o AS. Permitir que usuários finais acessem essas informações é uma forma que as entidades administradoras dos ASes empregam para mostrar ao público o crescimento da rede, servindo como mecanismo de publicidade.

Para permitir ao *mashup* criado exibir diferentes informações de acordo com o nível de acesso de cada usuário, foi implementado no NMMS um mecanismo de controle de fluxo de informação (*Information Flow Control* - IFC) (CHENG, 2009). Esse mecanismo é baseado em *labels*, que consistem em conjuntos de *tags* definidas pelo desenvolvedor de *mashups* para representar permissões arbitrárias. Durante a criação de um *mashup*, esse desenvolvedor pode marcar fluxos de informação dentro da sua composição com essas *tags*, definindo dessa forma que somente usuários com as permissões descritas pelas *tags* marcadas podem acessar aquelas informações.

Durante a execução do *mashup* marcado, *tags* são removidas de acordo com as permissões do usuário que solicitou a execução. Essa remoção é feita por uma extensão da *engine* de composição do NMMS implementada para avaliação de *tags*. Essa *engine* é responsável, também, por bloquear as informações que não tiveram suas *tags* removidas (*i.e.*, informações às quais o usuário não tem acesso).

Para criar *labels* para seus fluxos de informação, o desenvolvedor de *mashups* deve utilizar elementos de interação de confidencialidade disponíveis no ambiente de desenvolvimento do sistema de *mashups*. No NMMS, esses elementos foram materializados nos seguintes módulos:

- ***Add Tag*:** Permite que o desenvolvedor defina que determinado fluxo de informações requer *tags* específicas para ser acessado. Esse módulo recebe como entrada a saída de outro módulo da composição e um conjunto de *tags*. Ao conectar um *Add Tag* após um módulo do *mashup*, o desenvolvedor define que as informações retornadas por esse módulo só podem ser acessadas por usuários com permissões para remover as tags passadas como entrada;

- ***Remove Tag*:** Permite que o desenvolvedor defina quando uma *tag* pode ser removida dentro do *workflow* da sua composição, ou seja, quando os *labels* serão avaliados pela *engine* de composição para definir quais informações serão exibidas para o usuário do *mashup*. Extensões do módulo *Remove Tag* podem ser criadas para que operações sejam realizadas na remoção das *tags*, como, por exemplo, anonimização de endereços IP.

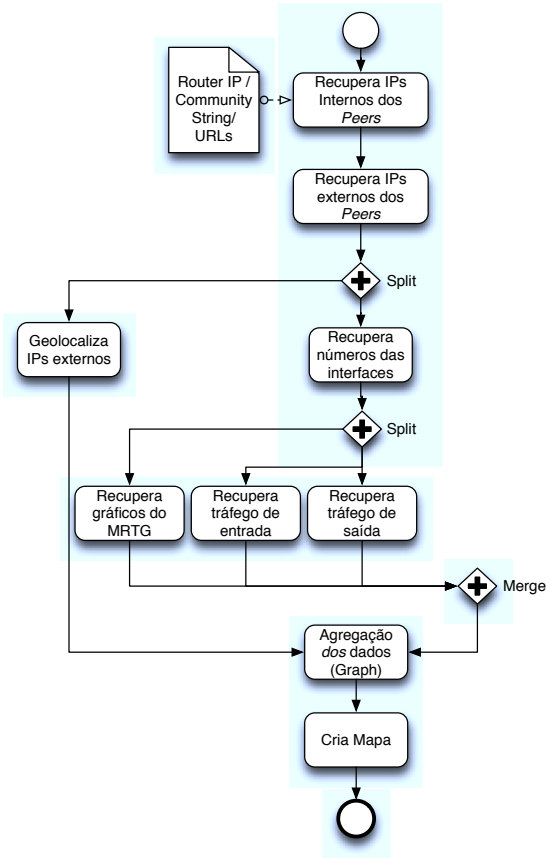

Figura 5.8: Resumo do *workflow* original do *mashup* de *peering* BGP

O *mashup Network Traffic Monitoring* exibe um mapa do *backbone* do AS gerenciado mostrando, de acordo com o nível de permissão do usuário, os PTTs, as conexões desses com clientes e as informações de tráfego adequadas. Ele foi criado com base na composição do *mashup* original de *peering* BGP, cujo *workflow* está representado na figura 5.8 para fins de referência. Como esse *mashup* original já suportava o caso de múltiplos roteadores BGP, foi necessário apenas alterar o módulo *Graph* para que o mesmo associasse um roteador de borda BGP a cada PTT. Já para implementar o controle de fluxo de informação no *Network Traffic Monitoring*, a lógica adotada está representada na figura 5.9. Na figura 5.9-A, quando as informações são recuperadas dos roteadores locais

dos PTTs, elas são marcadas com *tags* identificando sua confidencialidade. No *Network Traffic Monitoring*, as seguintes tags são inseridas:

- Duas *tags* para cada PTT: PTT-Detalhado (PTT_D) e AS-Detalhado (AS_D)
- Duas *tags* após agregar as informações: PTT-Agregado (PTT_A) e AS-Agregado (AS_A)

A figura 5.9-B mostra, por sua vez, que a remoção de *tags* é realizada antes das informações serem submetidas para o módulo que irá gerar a visualização do *mashup* (*i.e.*, Criar Mapa). Ao executar o módulo *Remove Tags* durante execução do *mashup*, a *engine* de composição irá remover tags de acordo com as permissões do usuário que requisitou a execução e submeterá para o módulo de criação de mapas apenas as informações às quais esse usuário tem acesso. Esse módulo, por sua vez, criará o mapa com base nessas informações, que são corretas em relação às permissões do usuário.

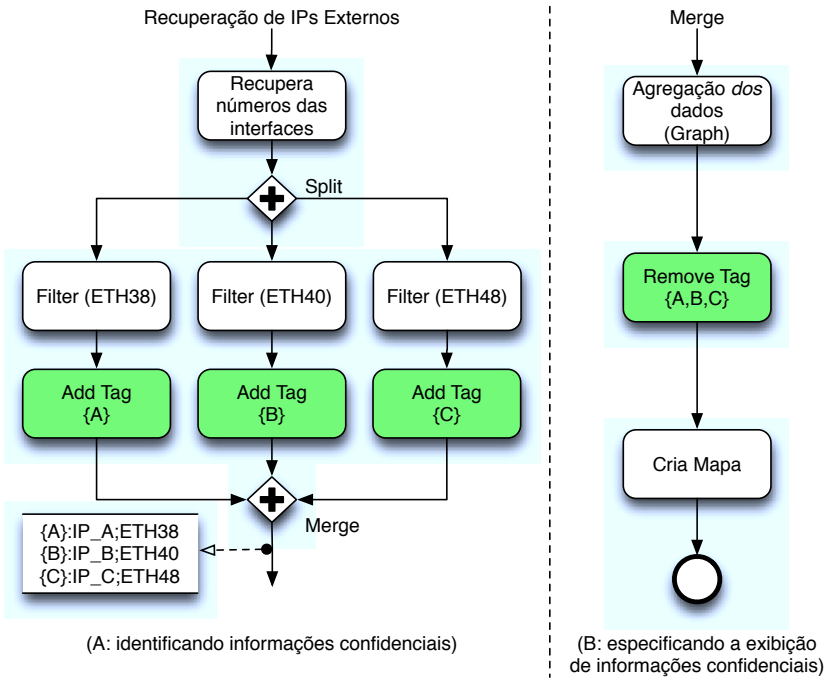

Figura 5.9: Lógica de controle de fluxo de informação no *Network Traffic Monitoring*

Mapas criados na execução do *Network Traffic Monitoring* para os diferentes níveis de acesso às informações de tráfego podem ser vistos na figura 5.10. Nesses mapas, marcadores verdes representam os pontos de troca de tráfego e as conexões entre os mesmos são representadas por conectores pretos. Os marcadores e conectores azuis, por sua vez, representam os clientes e suas conexões com os PTTs. Um clique em um marcador mostra todo o tráfego trocado pelo PTT ou com determinado cliente, dependendo do tipo de marcador. Já clicando em um conector azul, o usuário pode visualizar o tráfego trocado na conexão entre um cliente e um PTT.

A figura 5.10-A mostra a visão do CO, ou seja, todos os PTTs do *backbone* conectados com todos os clientes. Já a figura 5.10-B mostra a visão do administrador de um PTT, onde esse pode ver apenas o tráfego de clientes conectados ao seu PTT e o tráfego agregado dos demais pontos de troca do AS gerenciado. Finalmente, a figura 5.10-C apresenta a visão

do cliente, que só pode visualizar conexão e tráfego trocado com os PTTs aos quais ele está conectado. A visualização do usuário final foi omitida da figura por sua simplicidade. Ela compreende uma listagem do tráfego total do AS e, caso desejado pelo desenvolvedor do *mashup*, o *mapa* de PTTs do AS gerenciado.

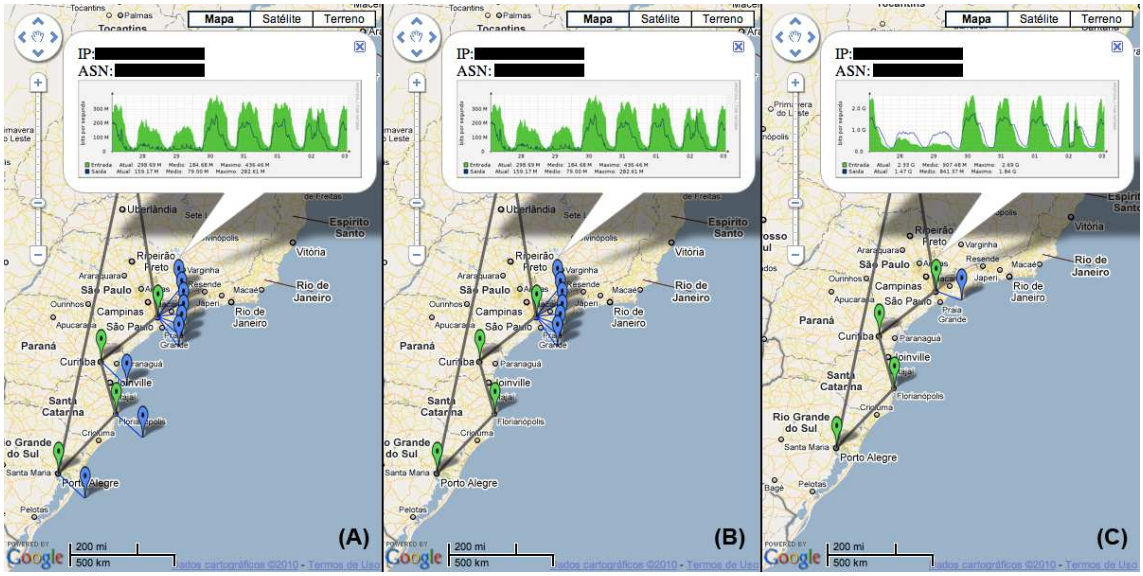

Figura 5.10: O *mashup Network Traffic Monitoring*: Visualizações distintas para diferentes níveis de acesso às informações

O *mashup* de *peering* BGP original abordava apenas um único PTT, atendendo às necessidades específicas do administrador do mesmo. Devido às capacidades de edição fornecidas pelo NMMS, esse *mashup* foi facilmente expandido para o caso de um AS com diversos PTTs, dando origem ao *Network Traffic Monitoring*. A implementação desse último, por sua vez, apresentou uma solução simples para o problema complexo de exibir informações distintas para usuários com diferentes interesses e permissões de acesso. A simplicidade dessa solução pode ser observada ao constatar que foram necessários implementar apenas dois novos módulos (*i.e.*, *Add Tag* e *Remove Tag*) e um mecanismo de avaliação de tags na *engine* de composição. Como esses módulos podem ser utilizados pelo administrador de redes através da linguagem visual do NMMS, adequada a usuários leigos em desenvolvimento, a solução para o problema de diferentes níveis de acesso à informação também apresenta usabilidade adequada.

A presente seção discutiu em detalhes o estudo de caso de *peering* BGP, sua solução utilizando um *mashup* criado no NMMS e uma expansão desse estudo de caso também resolvida utilizando o sistema. A próxima seção do capítulo, por sua vez, tratará do segundo estudo de caso da aplicação de *mashups* no gerenciamento de redes, que consiste no uso de *mashups* para detecção de *botnets*.

5.2 Estudo de Caso 2 - Detecção de *Botnets*

Atualmente, criadores de vírus de computador estão mais interessados em obter ganhos financeiros com suas criações do que em desabilitar as máquinas infectadas. Uma forma comum de alcançar esse tipo de ganho consiste em obter controle remoto de computadores infectados, transformando-os em *bots*. Esses *bots* podem ser, então, organizados em redes globais altamente heterogêneas denominadas *botnets*. Tais redes são compostas

de milhares de máquinas infectadas, que podem ser controladas remotamente por um operador humano denominado *botmaster* (WANG et al., 2010). Elas são normalmente usadas para propósitos ilegais como, por exemplo, ataques de negação de serviço distribuídos (*Distributed Denial of Service* - DDoS) massivos e *spamming* (PAUL BACHER THORSTEN HOLZ; WICHERSKI, 2005). Por serem compostas de milhares de computadores em diversos países e conectados a diferentes ISPs, as *botnets* são uma ameaça extremamente resiliente.

Os *botmasters* enviam comandos para suas *botnets* através de infraestruturas tecnológicas denominadas Canais de Comando e Controle (*Command and Control Channels* - C&C *Channels*). Esses canais podem ser implementados através de diversas tecnologias (DASWANI; STOPPELMAN, 2007) como, por exemplo, *Internet Relay Chat* (IRC), protocolos *peer-to-peer* (P2P) e mesmo ferramentas online recentes como o Twitter[1]. O conjunto de tecnologias que podem ser utilizadas em canais C&C está em constante evolução, à medida que os criadores de *malware* encontram novos métodos e tecnologias que permitem que comandos para os *bots* penetrem diferentes topologias de rede e medidas de segurança (*e.g., firewalls*).

Historicamente, os canais C&C das primeiras *botnets* adotavam uma arquitetura centralizada baseada em IRC (GOEBEL; HOLZ, 2007), a qual ainda é utilizada por muitas das *botnets* atuais. Esses canais de comando e controle centralizados, apesar de efetivos, têm um ponto central de falha por definição, o qual consiste no servidor C&C. Como os *bots* devem conectar-se com esse servidor para receber ordens do *botmaster*, desabilitar tal servidor ou impedir o contato com o mesmo são medidas que, efetivamente, desativam a *botnet*. Mesmo em arquiteturas de C&C distribuídas, identificar e bloquear o canal de comando e controle, isolando o *botmaster* dos *bots*, é uma das principais estratégias para combater uma *botnet*.

A estratégia de detecção de *botnets* utilizada no presente estudo de caso é baseada no trabalho descrito em (CERON; GRANVILLE; TAROUCO, 2010). Nesse trabalho, é definida uma arquitetura para descoberta de informações sobre *botnets* através da coleta e análise de binários infectados por *malware*. Esses binários, denominados *vetores*, consistem em executáveis maliciosos que servem para comprometer a máquina do usuário, inserindo-a em uma determinada *botnet*. A análise desses binários proposta em (CERON; GRANVILLE; TAROUCO, 2010) consiste em uma correlação do resultado de diversas ferramentas de análise de binários distintas disponíveis na Web. A maioria dessas ferramentas utiliza uma técnica de análise de *software* denominada *Sandboxing*, que consiste em observar o comporamento de um determinado binário executado em um ambiente controlado (*i.e.,* um *sandbox*), sumarizando em um relatório informações sobre o funcionamento desse binário e sobre as comunicações de rede realizadas pelo mesmo (WILLEMS; HOLZ; FREILING, 2007). A arquitetura proposta por (CERON; GRANVILLE; TAROUCO, 2010) contém um componente responsável por coletar esse tipo de relatório de diversas ferramentas, correlacionar esses relatórios e armazenar o resultado dessa correlação em um banco de dados de *botnets*. Nessa correlação, são empregadas heurísticas para descobrir, dentre outras informações, o endereço IP do servidor C&C com o qual o vetor analisado se comunica.

O *mashup* criado para o presente estudo de caso será abordado na próxima subseção. Ele é baseado na estratégia de detecção descrita acima, expandindo as informações obtidas através do uso de ferramentas disponíveis na Web para identificar a localização geográfica dos servidores C&C descobertos e qual o sistema autônomo responsável pelo

[1] www.twitter.com

endereço IP desses servidores.

5.2.1 *Mashup* Desenvolvido

O *mashup* de detecção de *botnets* criado utiliza as informações obtidas pela coleta e análise realizada pela arquitetura descrita em (CERON; GRANVILLE; TAROUCO, 2010) para criar um mapa interativo de servidores de comando e controle descobertos. Para esse fim, foi criado na arquitetura de detecção um serviço que, ao ser invocado, recupera as informações disponíveis no banco de dados de *botnets* e retorna essas informações no formato CSV. Tais informações incluem:

- *Hash* MD5 única de cada binário analisado;
- Data de submissão/coleta do binário;
- Tipo de *malware* detectado no binário analisado (*e.g.*, vírus, *worm*, *trojan*);
- Endereço IP do servidor C&C com o qual o *malware* se comunica ao infectar uma máquina;

O *mashup* criado compõe essas informações com serviços de geolocalização, *Whois* e mapas *online*. Inicialmente, essa subseção discutirá como essa composição foi realizada utilizando o NMMS e, a seguir, o *mashup* em si será apresentado e discutido.

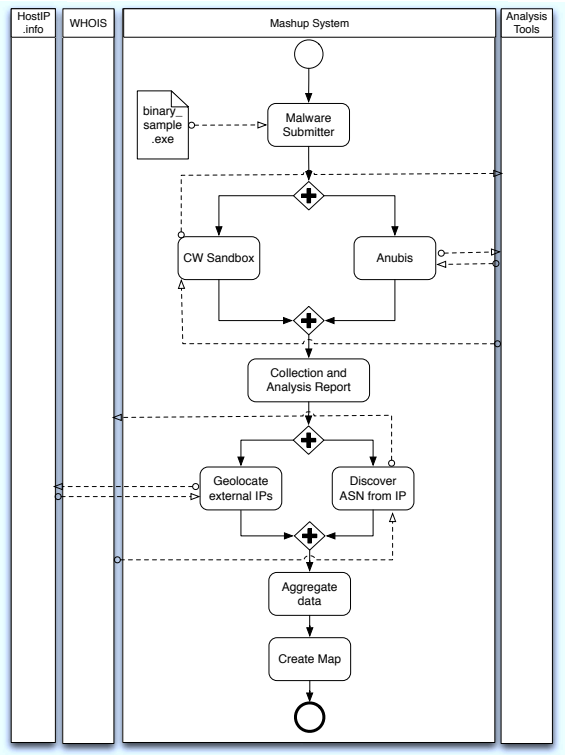

Figura 5.11: O *workflow* do *mashup* de detecção de *botnets*

O *workflow* do *mashup* de detecção de *botnets* pode ser visto na figura 5.11. Os módulos do NMMS que participam desse *workflow* são os seguintes:

- ***Analysis***: Módulo de adaptação para o serviço de informações sobre *botnets* descrito anteriormente;

- ***Whois***: Módulo de adaptação para uma ferramenta Web de *Whois*[1] capaz de, dado determinado endereço IP, recuperar a qual sistema autônomo ele pertence;

- ***Geolocation***: Módulo de geolocalização de IP, que já foi abordado no estudo de caso anterior;

- ***Aggregate***: Módulo de reuso responsável pela agregação das informações de *botnets*, geolocalização e *Whois*. Recebe essas informações como entrada e retorna uma lista de pontos geográficos representando as prováveis localizações de servidores C&C;

- ***Maps***: Módulo responsável pela criação de mapas interativos, descrito no estudo de caso anterior.

O processo de obtenção e análise de binários visto no *workflow* da figura 5.11 é abstraído pelo módulo *Analysis* e ocorre de maneira assíncrona à execução do *mashup* em si. Esse processo é responsabilidade de uma instância da arquitetura de detecção de *botnets* abordada na seção anterior, a qual irá coletar binários através de diferentes meios (*e.g.*, formulário de submissão, *honeypots*, análise de *spams*) e submeter esses binários para diferentes serviços de análise encontrados na Web. Os serviços de análise de binários utilizados no estudo de caso são os seguintes:

- CwSandbox[2] e Anubis[3]: Ambos são serviços que executam o binário infectado em um *sandbox* e analisam o comportamento do mesmo, identificando, por exemplo, comunicação com servidores C&C;

- VirusTotal[4]: Esse serviço analisa o binário submetido em busca de assinaturas de vírus, *trojans* e *worms*.

Após algumas horas, esses serviços disponibilizam relatórios contendo as informações descobertas sobre o binário analisado. Quando disponíveis, esses relatórios são coletados, processados e correlacionados pelo componente de coleta e análise da arquitetura de detecção. O resultado dessa correlação é armazenado em um banco de dados de *botnets*. Ao ser invocado, o serviço adaptado pelo módulo *Analysis* retorna um *snapshot* dessa base de dados, contendo todas as informações sobre *botnets* e servidores C&C colhidas até o momento da invocação.

A composição criada no NMMS, que pode ser vista na figura 5.12, tem como ponto de partida a execução do módulo *Analysis*. Os endereços IP dos servidores C&C que esse módulo recupera da base de dados de *botnets* são submetidos para o módulo *Geolocation*, o qual irá localizar esses servidores geograficamente em termos de latitude e longitude. Tais endereços IP também são submetidos para o módulo *Whois* que, por sua vez, identificará os ASN dos sistemas autônomos nos quais os servidores de comando e controle estão conectados. As saídas dos módulos *Analysis*, *Whois* e *Geolocation* são, então, submetidas para o módulo *Aggregate*, o qual irá gerar a lista de pontos geográficos correspondentes ao servidores C&C. Finalmente, essa lista é utilizada pelo *Maps* para gerar a visualização do *mashup*, marcando cada um dos pontos no mapa e anotando-os com as informações relativas aos mesmos.

[1] http://www.team-cymru.org/Services/ip-to-asn.html
[2] www.threattrack.com
[3] anubis.iseclab.org
[4] www.virustotal.com

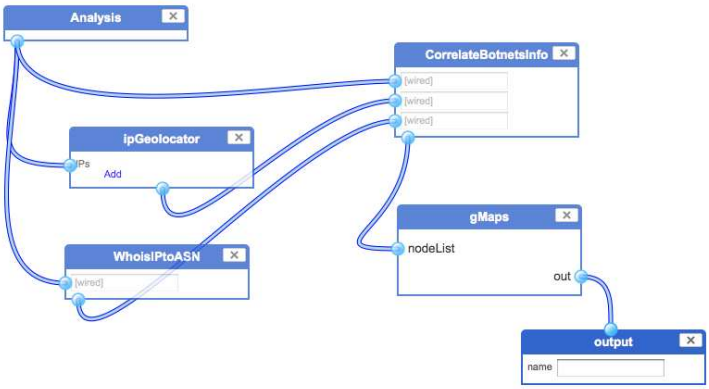

Figura 5.12: *Mashup* de detecção de *botnets*: Implementação no NMMS

O *mashup* de detecção de *botnets*, que pode ser visto na figura 5.13, fornece uma visão centralizada da distribuição mundial de servidores C&C. Nele, tais servidores são representados por pontos no mapa. Esses pontos possuem ícones diferentes dependendo do tipo de *malware* (*e.g.*, vírus, *worm*, *trojan*) que se comunica com cada servidor. Ao clicar em um ponto, pode-se visualizar informações relativas ao servidor C&C (*e.g.*, endereço IP, latitude, longitude, AS ao qual esse servidor se conecta) e ao binário que se comunica com o mesmo (*e.g.*, *hash* MD5, tipo de *malware*, data de coleta). Ao possibilitar a observação da concentração de servidores C&C em determinadas localidades e/ou ASes, o *mashup* criado dá indícios de onde concentrar esforços investigativos e quais entidades notificar para combater as *botnets* detectadas.

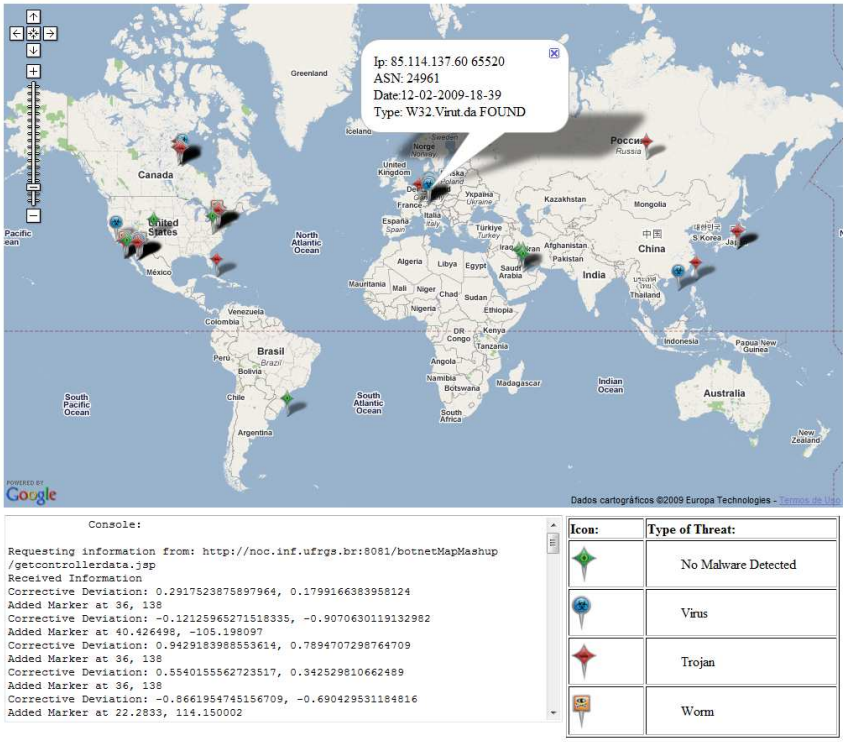

Figura 5.13: O *mashup* de detecção de *botnets*

Nessa subseção, foram discutidos o *mashup* de detecção de *botnets* e sua implementação no NMMS. A subseção seguinte analisará tanto essa implementação, comparando-a com o desenvolvimento do *mashup* de forma *ad-hoc*, quanto os resultados obtidos por esse *mashup*.

5.2.2 Discussão dos Resultados

O desenvolvimento *ad hoc* do *mashup* de detecção de botnets, realizado para fins de comparação com criação do mesmo através do NMMS, demandou o domínio das seguintes tecnologias:

- Linguagens de programação: Java e JavaScript;
- Formato de dados: CSV e JSON;
- Protocolos: HTTP e DNS;
- APIs *online*: APIs de geolocalização, API do Google Maps,

Assim como no *mashup* criado via NMMS, essa implementação utilizou como base uma instância da arquitetura de detecção de *botnets* proposta em (CERON; GRANVILLE; TAROUCO, 2010), incluindo o serviço que expõe as informações do banco de dados da mesma descrito na subseção anterior.

Os resultados da comparação das diferentes abordagens de desenvolvimento do *mashup* (*i.e.*, *ad hoc* e através do NMMS) são bastante similares aos observados no estudo de caso anterior. Assim como no caso do *mashup* de *peering* BGP, a implementação do *mashup* de detecção de *botnets* via NMMS mitigou a necessidade do domínio das diversas tecnologias listadas acima, as quais foram necessárias para o desenvolvimento *ad hoc*. Tal mitigação foi possível devido ao uso dos *wrappers* do NMMS, que se responsabilizaram pelas questões técnicas envolvidas no uso dessas tecnologias, permitindo ao desenvolvedor criar todo o *mashup* utilizando apenas módulos da camada de apresentação do sistema.

A comparação da manipulação de recursos externos no NMMS em relação à abordagem *ad hoc* no presente estudo de caso também apresenta resultados similares ao estudo anterior. Por exemplo, o trecho de código Java da figura 5.14 corresponde às operações necessárias para o *mashup ad hoc* descobrir a quais ASes pertencem os endereços IP dos servidores C&C. Tal descoberta é realizada através do serviço de *Whois* denominado *IP to ASN Mapping*, disponibilizado pela organização *Team Cymru*[1] através do protocolo DNS. Para o uso desse protocolo na linguagem Java, foi necessário empregar da biblioteca *dnsjava*[2]. A interação com esse mesmo *Whois* no desenvolvimento via NMMS está destacada na figura 5.15. Pode-se observar que a utilização do NMMS simplificou essa interação, resumindo as várias operações do código complexo visto na figura 5.14 à interação com um único módulo na interface de composição do sistema.

Analisando o *mashup* criado em si, pode-se perceber que o mesmo é uma alternativa para ferramentas comerciais como, por exemplo, a versão comercial do CWSandbox[3], diferentemente do *mashup* do estudo de caso de *peering* BGP, o qual atende uma necessidade não resolvida por ferramentas existentes. O *mashup* de detecção de *botnets*, no entanto, não tem como objetivo substituir as ferramentas existentes, mas sim utilizá-las

[1] http://www.team-cymru.org/Services/ip-to-asn.html
[2] http://www.xbill.org/dnsjava/
[3] http://www.gfi.com/pages/prodinfo.htm

```
1       public static Message queryAsnByIp(String address)
2                               throws UnknownHostException, TextParseException,
                                    IOException {
3               Message query, response;
4               String msg;
5               String ip = reverseIP(address);
6
7               if (ip.startsWith("\\.")) ip = ip.substring(1);
8               if (ip.endsWith("\\."))
9                       msg = ip+"origin.asn.cymru.com";
10              else
11                      msg = ip+".origin.asn.cymru.com";
12
13              Record rec;
14              SimpleResolver res;
15              res = new SimpleResolver();
16              rec = Record.newRecord(Name.fromString(msg, Name.root), Type.TXT, DClass
                    .IN);
17              query = Message.newQuery(rec);
18              response = res.send(query);
19              return response;
20      }
21
22      public static String extractAsnFromResponse(String response) {
23              Pattern as = Pattern.compile("\"\\d*\\d\\s|\"");
24              Matcher matcher = as.matcher(response);
25              String result;
26              if (matcher.find()){
27                      result = matcher.group(0);
28              } else result = "NotFound";
29
30              if (result.startsWith("\"")){
31                      result = result.substring(1);
32              }
33              return result;
34      }
```

Figura 5.14: Código Java para descobrir o AS ao qual determinado endereço IP pertence

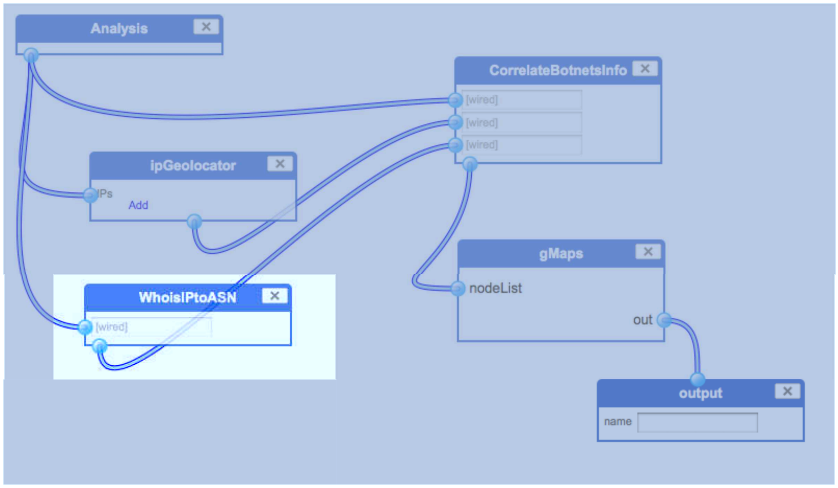

Figura 5.15: Obtenção de informação de *Whois* no *mashup* de detecção de *botnets* implementado no NMMS

de forma integrada. Ou seja, nada impede que a composição seja adaptada para utilizar *softwares* comerciais de detecção de *botnets*.

A adaptabilidade do *mashup* criado também é útil para permitir que o mesmo acompanhe a evolução constante das tecnologias utilizadas em canais C&C. Isso pode ser al-

cançado através da integração de ferramentas de detecção específicas para tipos de canais C&C distintos à composição. Por exemplo, pode ser possível integrar um software de detecção de canais C&C baseados em protocolos P2P à arquitetura de coleta e análise de binários, submetendo dados descobertos por esse *software* para a base de dados de *botnets*.

A extensibilidade é outro aspecto importante do *mashup* de detecção de *botnets*. Através da adição de outros módulos após a etapa de agregação dos dados, esse *mashup* pode ser estendido para outros fins além da detecção de *botnets*. Por exemplo, um módulo de criação de regras de *firewall* pode ser usado para criar regras que bloqueiem os endereços IP dos servidores C&C detectados, efetivamente impedindo que máquinas comprometidas dentro da rede sejam controladas por comandos gerados por esses servidores (DOS SANTOS et al., 2010). Além disso, a inclusão de um elemento de interação do tipo *Daemon* na composição possibilita, por exemplo, a criação de um *mashup* capaz de analisar fluxos de rede (*i.e., netflows*) continuamente, buscando identificar *bots* dentro da rede gerenciada que se comunicam com servidores C&C detectados. Finalmente, as informações sobre *botnets* podem ser facilmente publicadas em diversos formatos como, por exemplo, XML, RSS ou JSON, através da utilização de *wrappers* para esses formatos de dados. Isso permite que essas informações sejam mais facilmente consumidas por outros *softwares* de gerência fora do ambiente do NMMS.

O segundo estudo de caso da aplicação de *mashups* no gerenciamento de redes foi discutido em detalhes nessa seção. A próxima seção, por sua vez, fará uma análise qualitativa dos *mashups* criados em ambos os estudos de caso.

5.3 Avaliação Qualitativa

Os *mashups* criados nos estudos de caso serão avaliados qualitativamente nessa seção. A avaliação abordará os dois *mashups* em conjunto devido às similaridades entre eles (*e.g.*, ambos são *mashups* de mapa para situações de gerenciamento de redes). Os critérios qualitativos adotados nela são os seguintes:

- **Esforço de implementação:** O esforço necessário para a criação dos *mashups* de gerenciamento;
- **Esforço de utilização:** O esforço necessário para utilizar os *mashups* implementados;
- **Confiabilidade:** O quão confiáveis são as composições criadas. Ou seja, caso surja uma necessidade atendida por essas composições, o quão o administrador pode confiar no uso das mesmas como ferramentas de gerenciamento;
- **Flexibilidade:** Capacidade de adaptação dos *mashups* criados para diferentes cenários relacionados à necessidade que os originou;
- **Extensibilidade:** Possibilidade de utilizar os *mashups* dos estudos de caso em outras composições e/ou em conjunto com outras ferramentas de gerenciamento.

Nos estudos de caso, observou-se que o **esforço de implementação** de um *mashup* de gerenciamento seguindo uma abordagem *ad hoc* é similar à criação de um *script*. Assim como no *scripting*, tal abordagem demanda habilidade em desenvolvimento de *software* e o domínio de tecnologias relacionadas, além do tempo e esforço necessários para criar

e testar código. Já no caso da utilização do NMMS para criar os *mashups*, a comparação realizada nos estudos de caso mostrou uma redução considerável no esforço de implementação em relação ao desenvolvimento *ad hoc*. O NMMS eliminou tanto a necessidade de habilidades em desenvolvimento de *software* quanto do domínio de tecnologias relacionadas. Essa eliminação foi possibilitada por componentes das diversas camadas da arquitetura do sistema. Na camada de adaptação, os *wrappers* automatizaram com sucesso as questões técnicas envolvidas no uso de recursos externos. A linguagem baseada em módulos da camada de composição, por sua vez, abstraiu diversas das operações envolvidas na interação entre esses recursos. Finalmente, os elementos de interação da camada de apresentação permitiram que as composições fossem descritas em uma linguagem visual simplificada, mais adequada ao usuário leigo do que sintaxes complexas de linguagens de desenvolvimento de *software*.

O **esforço de utilização** dos *mashups* criados corresponde àquele necessário para utilizar as interfaces visuais baseadas em mapa dos mesmos. Em ambos os estudos de caso, essas interfaces centralizam a visualização de dados obtidos em diversas fontes distintas e permitem que o administrador acesse, através de poucos cliques, informações em maior nível de detalhes. Como anteriormente discutido, a coleta manual dessas informações requer que o administrador interaja com as diversas ferramentas envolvidas, o que demanda tempo e domínio dessas ferramentas. Além disso, integrar essas informações é um problema complexo, visto que ferramentas de gerenciamento geralmente não são projetadas visando esse tipo de integração. Finalmente, o uso do NMMS traz uma vantagem adicional em termos de esforço de utilização, a independência de plataforma. Tanto esse sistema quanto as composições criadas nele são aplicações Web, que executam no navegador do administrador de redes, sendo nativamente multi-plataforma. Essa independência elimina problemas advindos do uso em conjunto de ferramentas de gerenciamento em sistemas operacionais distintos como, por exemplo, incompatibilidades entre diferentes esquemas de codificação de caracteres (*e.g.*, ASCII, UTF-8).

A **confiabilidade** é um aspecto qualitativo no qual os *mashups* criados apresentam limitações. Essas limitações são herdadas dos recursos externos utilizados nas composições, em especial recursos Web como serviços de geolocalização e APIs *online*. Tais recursos não fornecem quaisquer garantias de disponibilidade e podem, a qualquer momento, alterar suas interfaces ou mesmo ser desativados, efetivamente desabilitando *mashups* que os utilizem. Para mitigar essas limitações de confiabilidade, pode-se implementar a redudância desses recursos, empregando alternativas amplamente disponíveis na Web. Por exemplo, o *wrapper* de geolocalização pode ser adaptado para acessar diferentes serviços e utilizar o melhor resultado deles, descartando pares de coordenadas iguais a (0,0), que normalmente indicam que a base de dados de geolocalização não conseguiu determinar a latitude e longitude associadas a determinado endereço IP. No caso do *wrapper* de serviços de mapas, pode-se usar alternativas como o OpenStreetMap[1] ou o MapQuest[2].

A **flexibilidade** dos *mashups* criados é uma consequência da modularidade fornecida pelo NMMS, que torna simples substituir componentes nas composições criadas para adaptá-las a novas necessidades. Essas adaptações consistem essencialmente em interações com a interface de composição do sistema para adicionar ou substituir módulos no *mashup* sendo adaptado. Graças a tal simplicidade de adaptação, foram delineadas possíveis modificações nos *mashups* de ambos os estudos de caso. Além disso, no estudo de

[1] http://wiki.openstreetmap.org/wiki/API

[2] http://developer.mapquest.com/web/products/open

caso de *peering* BGP, a flexibilidade pôde ser observada na prática, visto que na expansão desse estudo de caso o *mashup* criado foi adaptado para um cenário mais abrangente.

Os *mashups* de ambos os estudos de caso apresentam **extensibilidade** suficiente para ser utilizados em outros contextos. Isso pode ser feito através do próprio NMMS, que trata cada uma das composições criadas como módulos os quais, por sua vez, podem servir de componentes em outros *mashups* de gerenciamento. Além disso, como discutido anteriormente, os *mashups* criados podem ser adaptados para expor os dados integrados em diferentes formatos (*i.e.*, CSV, XML) através do uso de *wrappers* para esses formatos. Esse tipo de adaptação possibilita que tais dados sejam consumidos por outras ferramentas de gerenciamento fora do ambiente do NMMS. Isso permite, por exemplo, que as informações de *peering* BGP ou de detecção de *botnets* integradas nos estudos de caso sejam utilizadas como *datasets* para a geração de visualizações de gerenciamento (SALVADOR et al., 2011).

A presente seção apresentou dois estudos de caso de aplicação de *mashups* no gerenciamento de redes. Através desses estudos de caso, foi possível observar como o NMMS possibilita a criação de *mashups* de gerenciamento de forma simplificada. Além disso, a avaliação qualitativa desses *mashups* evidenciou tanto os benefícios da tecnologia de *mashups* para o administrador de redes quanto as limitações de confiabilidade da mesma. Na próxima seção, serão discutidas as conclusões do presente trabalho, obtidas na implementação do NMMS, na elaboração da arquitetura que embasou tal sistema e nos estudos de caso.

6 CONCLUSÕES E TRABALHOS FUTUROS

Nessa dissertação foi definido e implementado um sistemas de *mashups* de gerenciamento de redes denominado *Network Management Mashup System* (NMMS). Tal sistema foi empregado na criação de composições de recursos em estudos de caso extraídos de problemas reais de gerenciamento. Para possibilitar e embasar a criação do sistema, foi definida uma arquitetura de referência para sistemas de *mashups* de gerenciamento de redes. Essa arquitetura de referência foi criada com base na análise de diversos sistemas de *mashup* existentes.

A arquitetura proposta descreve os principais componentes de um sistema de *mashups* de gerenciamento. Esses componentes são separados logicamente segundo o modelo de três camadas (ECKERSON, 1995), considerado apropriado para sistemas *Web*. Ela também aborda como o sistema atende às necessidades de administradores de rede interessados em criar *mashups* de gerenciamento (*i.e.*, desenvolvedores) ou apenas em utilizar composições existentes. Finalmente, a arquitetura define e categoriza os elementos de interação manipulados pelo desenvolvedor para criar suas composições. Essa categorização é importante por fornecer uma visão geral dos tipos de componentes disponíveis para a criação de um *mashup* de gerenciamento.

Os princípios da arquitetura e o modelo de três camadas adotado embasaram o desenvolvimento do protótipo de sistema de *mashups*. Os *wrappers* implementados na camada de adaptação desse sistema automatizam questões técnicas de recursos externos como, por exemplo, geração de código *JavaScript* para API mapas *online*. A camada de composição, por sua vez, define uma linguagem que permite a criação de *workflows* através da conexão de elementos lógicos denominados módulos. Esses módulos podem, por fim, ser compostos visualmente através da interface de composição da camada de apresentação do NMMS. Nos estudos de caso realizados durante a pesquisa, verificou-se que o sistema como um todo possibilita a criação, edição e o *reuso* de *mashups* de gerenciamento.

O primeiro desses estudos de caso trata da integração de informações relativas ao cenário de *peeering* BGP, que corresponde ao relacionamento entre diferentes sistemas autônomos (ASes) que compõem a internet. A visualização centralizada dessas informações é uma necessidade de administradores de rede para a qual, até a redação do presente trabalho, não há nenhuma ferramenta de gerenciamento específica. O *mashup* criado atende a essa necessidade, compondo tais informações em um *mashup* de mapa. Esse estudo de caso foi expandido para um cenário onde o sistema autônomo gerenciado é dividido em diversos pontos de troca de tráfego independentes. Devido à flexibilidade fornecida pelo NMMS, a composição original pôde ser adaptada para o cenário expandido. Nessa expansão, também foi implementado um mecanismo de controle de fluxo de informação (CHENG, 2009) no NMMS para possibilitar que *mashups* criados lidem com situações de gerenciamento nas quais diferentes usuários possuem níveis de acesso a informação

distintos.

O segundo estudo de caso discute a aplicação de *mashups* na detecção de servidores de comando e controle (C&C), através dos quais operadores humanos controlam ilegalmente redes globais de computadores infectados por *malwares*, denominadas *botnets*. O *mashup* criado nesse estudo de caso utiliza como base uma arquitetura de detecção de *botnets* existente (CERON; GRANVILLE; TAROUCO, 2010) e integra informações obtidas por ela com recursos disponíveis na Web como, por exemplo, serviço de geolocalização e *Whois*. Assim como no primeiro estudo de caso, o resultado dessa integração pode ser visualizado em um mapa interativo criado através de uma API *online*. Esse *mashup* serve como uma alternativa gratuita para ferramentas comerciais de detecção de *botnets*. Entretanto, ele pode ser facilmente adaptado para receber informações de outros *softwares* de detecção de botnets, incluindo tanto tais ferramentas comerciais quanto *softwares* que detectam canais C&C baseados em diferentes tecnologias (*e.g.*, protocolos *peer-to-peer*, HTTP). Essa adaptabilidade é uma evidência que a aplicação de *mashups* é adequada para problemas de gerenciamento cujos requisitos estão em constante mudança ou evolução.

Para fins de comparação, os *mashups* criados nos estudos de caso foram implementados tanto no NMMS quanto seguindo uma abordagem *ad hoc*. Essa abordagem *ad hoc* demandou habilidades em desenvolvimento de *software* e o domínio de tecnologias voltadas a desenvolvedores. Isso torna a implementação *ad hoc* de *mashups* de gerenciamento tão inadequada para administradores de rede quanto a criação de *scripts*. Já a utilização do NMMS eliminou a necessidade do domínio das tecnologias de desenvolvimento necessárias na abordagem *ad hoc*. Além disso, o sistema diminuiu a complexidade envolvida na interação entre recursos heterogêneos, através da utilização da linguagem baseada em interconexão de módulos da camada de composição. Finalmente, o ambiente de desenvolvimento da camada de apresentação do NMMS permitiu que as composições fossem criadas através da manipulação de blocos visuais, mais adequados para administradores leigos em desenvolvimento de *software* do que a sintaxe complexa das linguagens de programação empregadas na abordagem *ad hoc*.

A análise qualitativa dos *mashups* dos estudos de caso mostrou que, ainda que tais *mashups* tenham apresentado um baixo esforço de utilização, muitas das vantagens apresentadas pelos mesmos são decorrentes do emprego do NMMS. As principais dessas vantagens dizem respeito à redução do esforço de implementação e alteração das composições proporcionada pelo sistema. O baixo esforço de implementação permite que *mashups* de gerenciamento sejam criados para atender necessidades situacionais específicas de um administrador. Já um baixo esforço de alteração traz flexibilidade e extensibilidade às composições criadas, possibilitando que tais composições sejam adaptadas para atender novos requisitos ou utilizadas em múltiplos contextos distintos. Além de vantagens, a análise qualitativa também apontou uma limitação de confiabilidade dos *mashups* como ferramentas de gerenciamento. Essa limitação é herdada dos recursos externos, os quais geralmente não oferecem garantias de funcionamento. Ela pode ser mitigada através do emprego de redundância, possibilitado pela ampla gama de recursos alternativos disponíveis na Web.

Os estudos de caso mostraram que, de fato, é possível e vantajoso aplicar *mashups* no gerenciamento de redes. Esses estudos de caso também evidenciaram que, através do uso de um sistema de *mashups* como descrito pela arquitetura proposta, a composição de ferramentas de gerenciamento pode ser realizada por um administrador leigo em desenvolvimento de *software*. Ou seja, a aplicação da tecnologia de *mashups* não substitui essas ferramentas, mas serve para proporcionar uma maneira padronizada e simplificada

de utilizá-las de forma integrada.

O uso de um sistema de *mashups* aproxima o administrador de redes do processo de desenvolvimento de ferramentas de gerenciamento. No entanto, o sistema não dispensa a figura do desenvolvedor de *software*. Tais desenvolvedores são necessários para criar *wrappers* para os recursos que compõem os *mashups*, visto que essa criação requer habilidades em desenvolvimento de *software*. A aplicação de *mashups* otimiza a distribuição de responsabilidades nesse processo de desenvolvimento, permitindo que administradores cuidem da lógica de gerenciamento de redes das ferramentas criadas enquanto desenvolvedores de *software* responsabilizam-se por lidar com os problemas de desenvolvimento envolvidos na criação dos *wrappers*.

O trabalho de pesquisa descrito nessa dissertação dá os primeiros passos na investigação do tema do uso de *mashups* no gerenciamento de redes. Entretanto, esse trabalho não aborda todas as possibilidades de pesquisa do tema. A seção seguinte dá indícios dos próximos passos nessa investigação, delineando trabalhos futuros sobre a aplicação de *mashups* no gerenciamento de redes.

6.1 Trabalhos Futuros

A implementação de melhorias no protótipo NMMS é uma evolução natural dessa pesquisa. Essas melhorias podem abordar, por exemplo, a implementação de funcionalidades que atendem requisitos não funcionais do sistema de *mashups* (*e.g.*, controle de usuários, compartilhamento de *mashups* entre diferentes sistemas) ou a implementação de novos tipos de elementos de interação (*e.g.*, elementos do tipo *daemon*). O tratamento dos requisitos não funcionais também pode ser melhorado na arquitetura de referência. Atualmente, tais requisitos são tratados majoritariamente pelo componente da camada de composição denominado *Publisher*. Esse componente pode ser expandido através da criação de módulos específicos suas responsabilidades (*e.g.*, módulo de controle de usuários, módulo de compartilhamento de *mashups*). Durante essa expansão, pode-se avaliar também a possibilidade de transformar o *Publisher* em uma nova camada do sistema de *mashups*, caso seja identificada uma sobrecarga de responsabilidades no componente.

Uma oportunidade interessante de pesquisa sobre o tema de *mashups* no gerenciamento de redes é o estudo da *mashabilidade* (*Mashability*) (GOARANY; KULCZYCKI; BLAKE, 2010) de ferramentas de gerenciamento existentes. Nesse estudo, pode-se definir mashabilidade como a facilidade de utilização dessas ferramentas de forma integrada com outros recursos e em *mashups* de gerenciamento. Para avaliar mashabilidade, é necessário definir um modelo de avaliação dessa característica no contexto do gerenciamento de redes, levando em consideração critérios como, por exemplo, a capacidade da ferramenta avaliada exportar dados em diferentes formatos (*e.g.*, XML, JSON) ou de disponibilizar um *front end* Web que possa ser acessado por um *wrapper*. Esse modelo seria, então, utilizado para avaliar um conjunto de ferramentas de gerenciamento que pode incluir, por exemplo, o MRTG, ferramentas de análise de *flows* e *firewalls*. Tal estudo de mashabilidade pode dar indícios tanto de ferramentas de gerenciamento potencialmente úteis em novos *mashups* quanto de ferramentas facilmente adaptáveis através de *wrappers*. Além disso, ele trará indicativos de como essas ferramentas podem melhorar no que diz respeito à integrabilidade.

Outra oportunidade de pesquisa que apresenta potencial é a investigação de um modelo de composição de recursos específico para o gerenciamento de redes. O modelo de composição consiste na forma como o sistema de *mashups* representa os recursos uti-

lizados nas composições e as interligações entre os mesmos. A arquitetura descrita no presente trabalho é abstrata suficiente para comportar diferentes modelos de composição distintos e, portanto, delega a definição desse modelo para a implementação do sistema de *mashups*. O modelo adotado pelo NMMS foi suficiente para os propósitos da pesquisa, tendo sido aplicado com sucesso na criação de *mashups* de gerenciamento. Entretanto, não há garantias que o NMMS adota um modelo de composição otimizado para o uso por administradores de redes. Um estudo aprofundado de como o administrador enxerga os recursos que ele utiliza para criar suas composições e as conexões entre esses recursos pode dar origem a um novo modelo de composição mais adequado ao contexto de gerenciamento de redes. Esse novo modelo, por sua vez, potencialmente trará ganhos em termos de esforço de implementação e adaptação de *mashups* de gerenciamento. Os primeiros indícios para a criação do modelo de composição de gerenciamento possivelmente serão encontrados na análise de *scripts* criados por administradores de redes, na análise de linguagens de composição de recursos como BPEL (ANDREWS et al., 2003) e em trabalhos que tratam de composição de serviços no gerenciamento (VIANNA et al., 2007).

Novos estudos de caso da aplicação de *mashups* no gerenciamento também podem ser investigados. A principal forma de descobrir potenciais estudos de caso conhecida até então é através de entrevistas com administradores de redes, buscando problemas vividos no cotidiano dos mesmos que podem ser resolvidos com *mashups*. Além disso, a interação com administradores é central em possíveis novas avaliações do protótipo NMMS, outro trabalho futuro importante. Uma avaliação de usabilidade efetiva para o NMMS envolve observar e avaliar a interação de vários administradores com o sistema. Apesar dos resultados importantes a serem obtidos nesse tipo de avaliação, ela demanda uma grande quantidade de recursos humanos e computacionais (*e.g.*, tempo de dezenas de administradores de redes em produção, infraestrutura para coletar dados das interações desses administradores), os quais excederam o que havia disponível para a pesquisa que originou essa dissertação.

REFERÊNCIAS

ABOUAF, E. **Wireit - A JavaScript Wiring Library**. Disponível em: http://neyric.github.com/wireit/. Acesso em: abr. 2012.

AMER-YAHIA, S.; HALEVY, A. What does web 2.0 have to do with databases? In: VERY LARGE DATA BASES, 33. **Proceedings...** VLDB Endowment, 2007. p.1443–1443. (VLDB '07).

ANDREWS, T. et al. **BPEL4WS, Business Process Execution Language for Web Services Version 1.1**. [S.l.]: IBM, 2003.

ANDROUTSELLIS-THEOTOKIS, S.; SPINELLIS, D. A Survey of Peer-to-Peer Content Distribution Technologies. **ACM Comput. Surv.**, New York, NY, USA, v.36, n.4, p.335–371, 2004.

ANKOLEKAR, A. et al. The two cultures: mashing up web 2.0 and the semantic web. **Web Semantics: Science, Services and Agents on the World Wide Web**, [S.l.], v.6, n.1, p.70–75, February 2008.

BANERJEE, N.; DASGUPTA, K.; MUKHERJEA, S. Providing middleware support for the control and co-ordination of telecom mashups. In: MNCNA. **Anais...** ACM, 2007. p.11.

BATTISTA, G. D.; REFICE, T.; RIMONDINI, M. How to extract BGP peering information from the internet routing registry. In: SIGCOMM WORKSHOP ON MINING NETWORK DATA, MINENET '06, New York, NY, USA. **Anais...** ACM, 2006. p.317–322.

BEZERRA, R. S. et al. On the Feasibility of Web 2.0 Technologies for Network Management: a mashup-based approach. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, 12. **Anais...** [S.l.: s.n.], 2010. p.487–494.

BOUILLET, E. et al. MARIO: middleware for assembly and deployment of multi-platform flow-based applications. In: ACM/IFIP/USENIX INTERNATIONAL CONFERENCE ON MIDDLEWARE, 10, MIDDLEWARE '09, New York, NY, USA. **Anais...** Springer-Verlag New York: Inc., 2009. p.26:1–26:7. (Middleware '09).

CERON, J. et al. Uma Solução para Gerenciamento de BGP em Pontos de Troca de Tráfego Internet. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS, 27. **Anais...** [S.l.: s.n.], 2009. p.105–115.

CERON, J.; GRANVILLE, L. Z.; TAROUCO, L. M. R. Uma Arquitetura Baseada em Assinaturas para Mitigação de Botnets. In: SBSEG 2010: X SIMPÓSIO BRASILEIRO EM SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS. **Anais...** SBC - Sociedade Brasileira de Computação, 2010.

CHENG, W. W.-Y. **Information Flow for Secure Distributed Applications**. 2009. Ph.D. — MIT, Cambridge, MA, USA. Também publicado como Technical Report MIT-CSAIL-TR-2009-040.

CURBERA, F. et al. Unraveling the Web Services Web - An Introdution to SOAP, WSDL,and UDDI. **IEEE Internet Computing**, [S.l.], v.6, n.2, p.86–93, 2002.

DASWANI, N.; STOPPELMAN, M. The anatomy of Clickbot.A. In: FIRST CONFERENCE ON FIRST WORKSHOP ON HOT TOPICS IN UNDERSTANDING BOTNETS, HOTBOTS '07, Berkeley, CA, USA. **Anais...** USENIX Association, 2007. p.11–11.

DAYAL, U. et al. Data integration flows for business intelligence. In: INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY: ADVANCES IN DATABASE TECHNOLOGY, 12., New York, NY, USA. **Proceedings...** ACM, 2009. p.1–11. (EDBT '09).

DOS SANTOS, C. R. P. et al. On using mashups for composing network management applications. **IEEE Communications Magazine**, Piscataway, NJ, USA, v.48, p.112–122, December 2010.

ECKERSON, W. W. **Three Tier Client/Server Architecture**: achieving scalability, performance, and efficiency in client server applications. [S.l.]: Bentham Science, 1995.

ENNALS, R.; GAY, D. User-friendly functional programming for web mashups. In: ACM SIGPLAN INTERNATIONAL CONFERENCE ON FUNCTIONAL PROGRAMMING, ICFP '07, New York, NY, USA. **Anais...** ACM, 2007. p.223–234.

GARRET, J. J. **AJAX**: a new approach to web applications. Disponível em: http://www.adaptivepath.com/ideas/essays/archives/000385.php. Acesso em abr. 2012.

GIACCARDI, E.; FOGLI, D. Affective geographies: toward a richer cartographic semantics for the geospatial web. In: ADVANCED VISUAL INTERFACES, New York, NY, USA. **Proceedings...** ACM, 2008. p.173–180. (AVI '08).

GOARANY, K.; KULCZYCKI, G.; BLAKE, M. B. Mining social tags to predict mashup patterns. In: SEARCH AND MINING USER-GENERATED CONTENTS, 2., New York, NY, USA. **Proceedings...** ACM, 2010. p.71–78. (SMUC '10).

GOEBEL, J.; HOLZ, T. Rishi: identify bot contaminated hosts by irc nickname evaluation. In: HOTBOTS'07: PROCEEDINGS OF THE FIRST CONFERENCE ON FIRST WORKSHOP ON HOT TOPICS IN UNDERSTANDING BOTNETS, Berkeley, CA, USA. **Anais...** USENIX Association, 2007.

GOLDSZMIDT, G.; YEMINI, Y. Distributed Management by Delegation. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, 15. **Anais...** IEEE Computer Society, 1995.

GOOGLE. **Google Maps API**. Disponível em: http://code.google.com/apis/maps/. Acesso em abr. 2012.

HARRINGTON, D.; PRESUHN, R.; WIJNEN, B. **RFC 3411 - An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks**. Disponível em: http://www.ietf.org/rfc/rfc3411.txt?number=3411. Acesso em 06 abr. 2012.

HAWKINSON, J.; BATES, T. **Guidelines for creation, selection, and registration of an Autonomous System (AS)**. [S.l.]: IETF, 1996. n.1930. (Request for Comments).

HOYER, V. et al. Enterprise Mashups: design principles towards the long tail of user needs. In: IEEE SCC. **Anais...** IEEE Computer Society, 2008. p.601–602.

JSON. **Introducing JSON**. Disponível em: http://www.json.org/. Acesso em abr. 2012.

KAMOUN, F. Toward best maintenance practices in communications network management. **Int. J. Netw. Manag.**, New York, NY, USA, v.15, n.5, p.321–334, 2005.

KEITH, J. **DOM Scripting**: web design with javascript and the document object model. 1.ed. [S.l.]: friends of ED, 2005.

KOLLHOF, J.-K. **JSON-RPC Specifications**. Disponível em: http://json-rpc.org/wiki/specification. Acesso em: abr. 2012.

LEINWAND, A.; CONDROY, K. F. **Network Management**: a practical perspective. 2.ed. [S.l.]: Menlo Park, 1996.

LIU, H.; RAMASUBRAMANIAN, V.; SIRER, E. G. Client behavior and feed characteristics of RSS, a publish-subscribe system for web micronews. In: IMC '05: PROCEEDINGS OF THE 5TH ACM SIGCOMM CONFERENCE ON INTERNET MEASUREMENT, New York, NY, USA. **Anais...** ACM, 2005. p.1–6.

LIU, X. et al. Towards Service Composition Based on Mashup. In: SERVICES, 2007 IEEE CONGRESS ON. **Anais...** [S.l.: s.n.], 2007. p.332–339.

MAXIMILIEN, E. M.; RANABAHU, A.; TAI, S. Swashup: situational web applications mashups. In: OOPSLA '07: COMPANION TO THE 22ND ACM SIGPLAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING SYSTEMS AND APPLICATIONS COMPANION, New York, NY, USA. **Anais...** ACM, 2007. p.797–798.

MAXIMILIEN, E.; RANABAHU, A.; GOMADAM, K. An Online Platform for Web APIs and Service Mashups. **Internet Computing, IEEE**, [S.l.], v.12, n.5, p.32 –43, September 2008.

MCCLOGHRIE, K.; ROSE, M. T. **Management Information Base for Network Management of TCP/IP-based internets**:mib-ii. United States: RFC Editor, 1991.

MERELO-GUERVOS, J. J. et al. Asynchronous distributed genetic algorithms with Javascript and JSON. **2008 IEEE Congress on Evolutionary Computation IEEE World Congress on Computational Intelligence**, [S.l.], p.1372–1379, 2008.

MERRIL, D. **Mashups**: the new breed of web app - an introduction to mashups. Disponível em: http://www.ibm.com/developerworks/web/library/x-mashups.html. Acesso em abr. 2012.

MOHAMMADI, S.; KHALILI, A.; ASHOORI, S. Using an Enterprise Mashup Infrastructure for Just-in-Time Management of Situational Projects. **E-Business Engineering, IEEE International Conference on**, Los Alamitos, CA, USA, v.0, p.3–10, 2009.

MURUGESAN, S. Understanding Web 2.0. **IT Professional**, [S.l.], v.9, n.4, p.34 –41, july-aug. 2007.

NAKANO, Y. et al. Method of creating web services from web applications. In: IEEE INTERNATIONAL CONFERENCE ON SERVICE-ORIENTED COMPUTING AND APPLICATIONS, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2007. p.65–71.

NATURE. **Internet Encyclopaedias Go Head to Head**. Disponível em: http://www.nature.com/nature/journal/v438/n7070/full/438900a.html. Acesso em abr. 2012.

NIELSEN, J. **Usability Engineering (The Morgan Kaufmann Series in Interactive Technologies)**. [S.l.]: Morgan Kaufmann, 1994.

NOBRE, J.; GRANVILLE, L. Consistency maintenance of policy states in decentralized autonomic network management. In: NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (NOMS), 2010 IEEE. **Anais...** [S.l.: s.n.], 2010. p.519 –526.

OETIKER, T. **MRTG - The Multi Router Traffic Grapher**. Disponível em: http://oss.oetiker.ch/mrtg/. Acesso em jul. 2012.

O'REILLY, T. **What is Web 2.0**. Disponível em: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html. Acesso em abr. 2012.

PAUL BACHER THORSTEN HOLZ, M. K.; WICHERSKI, G. **Know your Enemy**:tracking botnets. [S.l.: s.n.], 2005. Disponível em: http://www.honeynet.org/papers/bots. Acesso em abr. 2012.

PAUTASSO, C.; ZIMMERMANN, O.; LEYMANN, F. Restful web services vs. b̈ig̈ web services: making the right architectural decision. In: WWW '08: PROCEEDING OF THE 17TH INTERNATIONAL CONFERENCE ON WORLD WIDE WEB, New York, NY, USA. **Anais...** ACM, 2008. p.805–814.

PROGRAMMABLEWEB. **ProgrammableWeb.com - Mashups, APIs and the Web as Plataform**. Disponível em: http://www.programmableweb.com. Acesso abr. 2012.

QINGCHENG, L.; YOUMENG, L. Extracting Content from Web Pages Based on RSS. In: INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND SOFTWARE ENGINEERING - VOLUME 05, 2008., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2008. p.218–221. (CSSE '08).

REKHTER, Y.; LI, T. **A Border Gateway Protocol 4 (BGP-4)**. United States: RFC Editor, 1995.

RODRIGUEZ, A. **RESTful Web services**: the basics. Disponível em: http://www.ibm.com/developerworks/webservices/library/ws-restful/index.html. Acesso em abr. 2012.

RUBIO, D. **An Introduction to JSON**. Disponível em: http://dev2dev.bea.com/pub/a/2007/
02/introduction-json.html. Acesso em abr. 2012.

SALVADOR, E. M. et al. A characterization study of SNMP usage patterns. In: INTEGRATED NETWORK MANAGEMENT. **Anais...** [S.l.: s.n.], 2011. p.690–693.

SANTOS, C. R. P. dos et al. A data confidentiality architecture for developing management mashups. In: INTEGRATED NETWORK MANAGEMENT. **Anais...** [S.l.: s.n.], 2011. p.49–56.

SCHONWALDER, J. et al. SNMP Traffic Analysis: approaches, tools, and first results. In: INTEGRATED NETWORK MANAGEMENT, 2007. IM '07. 10TH IFIP/IEEE INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2007. p.323 –332.

SCHONWALDER, J. et al. Future Internet = content + services + management. **Communications Magazine, IEEE**, [S.l.], v.47, n.7, p.27 –33, 7 2009.

SCHONWALDER, J.; QUITTEK, J.; KAPPLER, C. Building distributed management applications with the IETF Script MIB. **Selected Areas in Communications, IEEE Journal on**, [S.l.], v.18, n.5, p.702–714, 2000.

SILVER, B. **BPMN Method and Style**: a levels-based methodology for bpm process modeling and improvement using bpmn 2.0. [S.l.]: Cody-Cassidy Press, 2009.

SOLDATOS, J.; ALEXOPOULOS, D. Web services-based network management: approaches and the wsnet system. **Int. J. Netw. Manag.**, New York, NY, USA, v.17, n.1, p.33–50, 2007.

SOMMERVILLE, I. **Software Engineering**. [S.l.]: Addison Wesley, 2004.

THOMAS, J. Rich Internet Applications (RIA): ajax, ruby on rails, and php: tutorial presentation. **J. Comput. Sci. Coll.**, USA, v.23, p.164–164, June 2008.

THOR, A.; AUMUELLER, D.; RAHM, E. Data Integration Support for Mashups. In: SIXTH INTERNATIONAL WORKSHOP ON INFORMATION INTEGRATION ON THE WEB, IIWEB, 2007. VANCOUVER, CANADA. **Proceedings...** AAAI Press, 2007.

TSAI, W. T. Service-oriented system engineering: a new paradigm. In: SERVICE-ORIENTED SYSTEM ENGINEERING, 2005. SOSE 2005. IEEE INTERNATIONAL WORKSHOP. **Anais...** [S.l.: s.n.], 2005. p.3–6.

TUCHINDA, R.; SZEKELY, P.; KNOBLOCK, C. A. Building Mashups by example. In: INTELLIGENT USER INTERFACES, 13., New York, NY, USA. **Proceedings...** ACM, 2008. p.139–148. (IUI '08).

VIANNA, R. L. et al. An Evaluation of Service Composition Technologies Applied to Network Management. In: INTEGRATED NETWORK MANAGEMENT. **Anais...** IEEE, 2007. p.420–428.

WANG, G. Improving Data Transmission in Web Applications via the Translation between XML and JSON. **Communications and Mobile Computing, International Conference on**, Los Alamitos, CA, USA, v.0, p.182–185, 2011.

WANG, P. et al. Honeypot detection in advanced botnet attacks. **Int. J. Inf. Comput. Secur.**, Inderscience Publishers, Geneva, SWITZERLAND, v.4, n.1, p.30–51, 2010.

WILLEMS, C.; HOLZ, T.; FREILING, F. Toward Automated Dynamic Malware Analysis Using CWSandbox. **IEEE Security and Privacy**, Piscataway, NJ, USA, v.5, n.2, p.32–39, 2007.

WILLIS, S.; BURRUSS, J.; CHU, J. **Definitions of Managed Objects for the Fourth Version of the Border Gateway Protocol (BGP-4) using SMIv2**. Obsoleted by RFC 4273, RFC 1657 (Draft Standard).

WINER, D. **XML-RPC Specification**. [S.l.]: UserLand Software, Inc., 1999.

WINETT, J. **Definition of a socket**. [S.l.]: IETF, 1971. n.147. (Request for Comments).

YU, J. et al. Understanding Mashup Development. **IEEE Internet Computing**, Piscataway, NJ, USA, v.12, n.5, p.44–52, 2008.

ZANG, N.; ROSSON, M. B.; NASSER, V. Mashups: who? what? why? In: CHI '08: CHI '08 EXTENDED ABSTRACTS ON HUMAN FACTORS IN COMPUTING SYSTEMS, New York, NY, USA. **Anais...** ACM, 2008. p.3171–3176.

APÊNDICE A ARTIGO PUBLICADO

Neste anexo é apresentado o artigo *"On the Feasibility of Web 2.0 Technologies for Network Management: A Mashup-Based Approach"*. O artigo foi desenvolvido na metade do segundo ano do mestrado e apresenta o trabalho de investigação de *mashups* no gerenciamento de redes descrito nessa dissertação. Esse artigo discute a arquitetura de referência para um sistema de *mashups* de gerenciamento proposta, apresenta o NMMS e, por fim, trata do estudo de caso de *Peering* BGP. Os resultados obtidos na avaliação realizada através do estudo de caso apontam a viabilidade de empregar *mashups* no gerenciamento de redes.

- **Título:** On the Feasibility of Web 2.0 Technologies for Network Management: A Mashup-Based Approach
- **Nome:** 12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010)
- **URL:** http://www.ieee-noms.org/
- **Data:** De 13 a 23 de abril de 2010
- **Local:** Osaka, Japão

On the Feasibility of Web 2.0 Technologies for Network Management: A Mashup-Based Approach

Rafael Santos Bezerra,
Carlos Raniery Paula dos Santos,
Leandro Marcio Bertholdo,
Lisandro Zambenedetti Granville,
Liane Rockenbach Tarouco
Institute of Informatics
Federal University of Rio Grande do Sul, Brazil
Email: {rsbezerra, crpsantos, granville}@inf.ufrgs.br, {berthold,liane}@penta.ufrgs.br

***Abstract*—Mashups are a new breed of Web applications, created through the integration of external resources available on the Web. Recently, they have been considered a hallmark of Web 2.0 technologies, placing the end user on a developer role and encouraging both collaboration and reuse. Following the increasing efforts in investigating new approaches to network management, mashups present themselves as a technology that can bring several advantages to the field. However, to this date, the usage of mashups in network management remains unexplored. Therefore, the present paper approaches this subject, proposing a Mashup Development Tool to network management. We discuss both the architecture of such system and a proof of concept prototype. We them employ our prototype to address the case study of integrating Autonomous System routing information.**

I. Introduction

The ever increasing complexity of computer networks constantly demands network management tools with more sophisticated capabilities. Usually, however, such capabilities do not come along with adequate user interfaces presented to the human network operators. Despite the current great availability of visual programming libraries and interactive solutions on the Web, many network management sytems (*e.g.*, Nagios, OpenNMS, OpManager) persist, requiring skilled human operators to run them. Besides, such solutions cannot address, for example, situations where managers need to visualize information in a specific and meaningful way, or when such information needs to be aggregated with external data from heterogeneous sources. This scenario is more problematic in the management of large backbones, because the management information usually comes from many different systems and devices.

Motivated by the aforementioned issues, there is a recent research trend towards the use, in the network management field, of technologies originally established and successfully employed in other areas. This trend includes, for example, the investigation of Web services [1] and *peer-to-peer* (P2P) [2] for network management: Web services traffic can cross administrative boundaries easier than SNMP because they use Internet protocols (*e.g.*, HTTP, FTP, and SMTP) as the underlying transport mechanism, while P2P potentially enables human-centric collaborative management along different administrative domains. Among the new technologies available, a set of novel ones, referenced with the title of Web 2.0, has been catching the attention of both industry and academy. Web 2.0 designates a new kind of Web applications where users are motivated to actively create and organize contents available on the Web [3].

In the myriad of technologies and applications that define the Web 2.0, one is of special interest in this paper: the *mashups*. Mashups are Web applications created from the composition and reuse of pre-existing external resources. Current work and approaches on mashups propose Web-based systems that allow end users with no programming expertise to develop their own applications. This development happens through the dynamic integration of existing resources like Web services, Web pages, and RSS feeds. This integration, which is inherently complex, is usually enabled by high-level abstractions and user-friendly interfaces [4]. As far as the authors of this paper are aware of, no effort towards the investigation of mashups for network management has been carried out so far, despite the existence of both academic [5] and industry [6] efforts on general purpose mashup development systems.

Considering the potential advantages of mashups and the lack of research about Web 2.0 technologies applied to network management, we aim in this paper at investigating whether the characteristics usually mentioned by mashup advocators hold when mashups are employed for network management. In particular, the following questions will be investigated:

- Can network management information be composed dynamically with external data from heterogeneous sources?
- How those compositions are better than traditional approaches used by network managers, such as script programming?
- Can mashups address network management situational needs?

In order to answer these questions, we have designed and implemented an architecture of a mashup system that enables the composition of management information sources available on the Web. As case study, we created, using the developed

system, a mashup to aggregate and monitor routing information collected from Autonomous Systems (ASes) that operate on the Internet using the Border Gateway Protocol (BGP). The aggregated information includes the amount of traffic exchanged between ASes and the number of routes announced by one AS to another, in a scenario known as BGP peering. Monitoring BGP peering is important because ASes rely on it to better maintain the health of Internet connectivity. The analysis of the aggregated information can lead to evidences of several critical problems. For example, it is possible for an AS to detect if a Service Level Agreement (SLA) is being broken by a neighbor Internet Service Provide (ISP).

The remainder of this paper is organized as follows. In Section 2 we review the main background and related work on the mashup technology. In Section 3 we introduce our proposed architecture that aims to support the development of mashups for network management. Also in Section 3, we describe the developed mashup sytem prototype. In Section 4 we discuss the case study. Finally, we draw conclusions and propose future work in Section 5.

II. Related Work

Mashups are Web applications created from the composition of preexisting Web resources like dynamic Web pages, Web services, and RSS feeds [5]. One of the main distinctions between mashups and other traditional composition technologies (*e.g.*, BPEL) is the goal of enabling average Web users with no programming skills to create their own applications, which better address their needs. Traditional composition technologies, on the other hand, require reasonably skilled developers that have sufficient knowledge about, for example, programming languages and paradigms, communication protocols, and distributed systems.

End user created mashups are built using mashup development systems, which themselves tend to be Web applications too. Usually, mashup development systems accumulate the roles of mashup repositories and runtime environments. This leads to the mixed scenario where a single system is used to create, store, and execute mashups. That does not mean that a mashup created in one system cannot be moved to be executed in another external system, or that it cannot be stored in a third system to form a remote library of mashups. Although this is all possible, in most of the current cases mashups are created, stored, and executed in the same environment. The term mashup system is usually employed to refer to such an environment. Mahsup systems are normally implemented using traditional Web technologies such as PHP and Java Server pages (JSP), and present sophisticated graphical Web interfaces created with usability-oriented technologies such as Asynchronous Javascript and XML (AJAX) [7] and Macromedia Flash.

The general components that compose the typical architecture of a mashup system are presented in Figure 1. Two different users interact with the mashup system: developers and end users. Developers define the compositions that result in the final mashups. End users, in their turn, are those interested in using the mashups created by developers. Once accessed by end users, developed mashups are executed to materialize the compositions previously specified. In this process, the mashup system retrieves information from remotely located Web resources (using technologies like SOAP, RPC, and Syndication protocols), performs operations (*e.g.*, sorting, filtering, aggregating) on the retrieved data, and finally builds a Web page that presents the composition results.

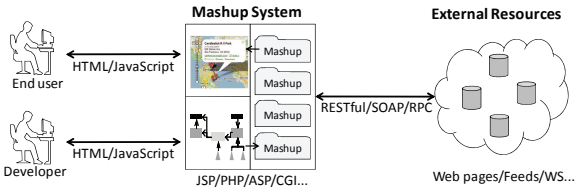

Fig. 1. Mashup system architecture

Although Figure 1 separates mashup developers and end users, both roles can be played by the same single person. In fact, the possibility of unskilled end users acting as developers is one of the key advantages advertised by mashup advocators. Whether such advantage really holds depends, among other factors, on the target problem the end user aims at solving and the mashup system used. Having that said, there are currently a set of relevant efforts on both academia and industry on researching and deploying mashup-based solutions.

One of the first investigations on mashups has been carried out by Banerjee *et al*. [8]. The authors have discussed the use of mashups in the development of telecommunication applications. They have proposed a multi-layered architecture that employs Web services as components, but with focus on easiness for inexperienced end users. The proposed architecture has been used, however, in the development of a single, very specific mashup (*i.e.*, call-a-cab) that does not allow, for example, an end user to build his/her own mashups. In addition, authors have not shown examples on how their architecture could be used in other telecommunication scenarios.

Yu *et al*. [9] have presented an overview on mashup development and the differences between mashups and traditional composition approaches. The authors have proposed a framework to classify different mashups systems in terms of what is provided by such systems, and how that is done. They also have identified some open research topics on mashups; for example, the authors observe that many of today's mashups still have a very limited audience, which may lead to possible scalability problems in future, more spread intensive usage. They also conclude that the improvement of mashup systems and the proliferation of components or modules for mashups are essential to help end users to build their own applications.

In the industry, one of the most relevant movements is related to building mashup development systems. Some examples are JackBe Presto[1], Yahoo! Pipes[2], IBM Mashup

[1] http://www.jackbe.com/presto
[2] http://pipes.yahoo.com

Center[3]. Pipes is almost exclusively related of integrating Web information through mashups. However, the other aforementioned examples are a different kind of effort, called enterprise mashups [10]. This proposal holds some similarities with SOA, proposing to develop an infrastructure of components and a system to allow dynamic and facilitated integration inside the context of an organization. Such efforts indicate that the industry believes in the potential advantages of the mashup technology.

The current industry initiatives and research efforts, as presented above, look at mashups usually considering specific scenarios. We believe that mashups can also be employed in the specific field of network management, with the potential advantage of allowing network human operators to create themselves their own, highly customized management applications without having to be trained, for example, in a complex composition technology. In the next section we introduce our mashup proposal in order to investigate this issue. We describe an architecture for a tool that aims to help network administrators to build their management mashups.

III. Proposal

To better understand the behavior of mashups when applied in network management, we defined an architecture for a mashup-based network management system. Based on this architecture, we developed a prototype tool that enabled us to create mashups and thus understand if they present advantages for the network management. It is important to note that, while this architecture describes a tool for network management, it is generic enough to describe even general purpose mashup systems.

A. Tiers

Figure 2 presents the architecture proposal and how its elements are related to each other. As aforementioned, we have two conceptual user roles, that can be, and ideally are, performed by the same user. The role of the developer relates to network operators interested in creating mashups for a specific network management need, and an administrator is one only interested in using the mashups already created by developers. With the proposed architecture, network operators acting as developers will be able to use their own knowledge to build applications more suitable to their needs. Besides, the architecture enables operators from different networks to collaborate with one another in creating more sophisticated management mashups. Another characteristic of the proposed architecture is that it allows mashup sharing between mashup systems, *i.e.*, a mashup created for a specific system can be accessed by other systems.

We designed the proposed architecture as a Web application based on the three-tier client-server model (*i.e.,* logic, presentation, data) [11]. By doing that, we could better separate the logical elements and individually handle them. For example, a change of external resources would only affect the implementation of the adaptation tier. These three tiers are:

- ***Adaptation Tier***: responsible for accessing and adapting external resources into a common format through elements called wrappers;
- ***Composition Tier***: orchestrates the execution of mashups, and performs operations (*e.g.*, sorting, filtering, aggregating) over the retrieved information;
- ***Presentation Tier***: where both end users, and network managers can create and/or use mashups, through the Development Environment (DE) and the Runtime Environment (RE).

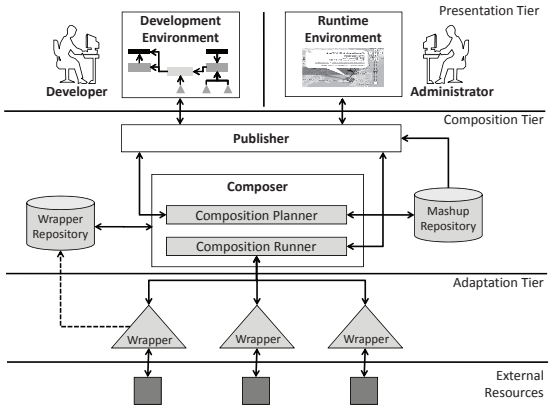

Fig. 2. Proposed architecture

B. Elements and Interactions

As mentioned before, wrappers are elements responsible for accessing external information, and making it available to the mashup system. In fact, they act as gateways to the different access methods (*e.g.*, SOAP, RPC) and data formats (*e.g.,* XML, CSV). For each wrapper, a set of meta-information describing the accessed resource is defined and stored in the Wrapper Repository. Examples of meta-information are input and output parameters, and resource description. This meta-information has two main goals: to allow the external resource to be accessed, and to describe resources to developers. The mashup system reads this meta-information and, based on it, developers can create their compositions. When a mashup is executed, each relevant wrapper for that mashup starts, retrieves external data, and forwards it to elements of the composition tier, which will integrate the retrieved data and build a Web page presenting the composition result.

On the composition tier, the Composer element is responsible for managing the compositions created by developers. When a mashup is created, a module of the Composer, called Composition Planner, receives the integration defined by developers (in the form of external resources pointers and relationships between these resources), validates such integration, transforms it into a common representation that can be processed by the mashup system, and finally stores it in the Mashup Repository. Another module of the Composer is the Composition Runner, which orchestrates calls to the

[3]http://www-01.ibm.com/software/info/mashup-center/

elements when executing the compositions stored by the Planner. The Composition Runner interacts directly with wrappers, processing the information received from them and outputting the mashup presentation (*i.e.*, a Web page).

Another element of the composition tier is the Publisher, whose function is to guarantee the access control of mashups. When a developer creates a mashup, he/she defines groups and roles. For example, a developer can define that some mashup cannot be edited, but it could be extended or used by the a specific group of users. The Publisher is also responsible for allowing mashups to be moved to external systems to be remotely executed. In this case, there are two mashup systems involved: the requester and the provider. The requester queries the provider's Publisher for a list of available mashups. The provider, in its turn, accesses its internal repository and replies with the list of mashups enabled to be externally executed. When the requester system asks for some specific mashup, both wrappers and mashup definition are delivered back.

The presentation tier has elements that are executed on the client-side, in a Web browser: the aforementioned Development Environment (DE) and the Runtime Environment (RE). The DE is used by the developer to create mashups by selecting which information will be integrated, setting the input data, and establishing relationships between them. The RE module, in its turn, exhibits the final results of the compositions (*i.e.*, mashups). When a network manager accesses the RE, the Publisher is called. It interacts with the Composition Runner requesting the execution of the selected mashup, receives the output (*e.g.*, HTML plus Javascript page) and sends it to the Presentation Layer, which will display the output to the user.

C. Interaction Components

In order to allow even unskilled network operators to build their own management applications, high-level abstractions should be used. These abstractions hide technical details of the external resources and of the operations executed on the retrieved data. To achieve that, we propose a set of interaction components that are displayed to network operators when acting as developers. These components allow the operators to define the business logic behind a mashup. Four types of components are proposed, although the architecture is extensible enough to support new ones. These four components are:

- **Visual**: represents the way results can be displayed, for example, in a map, table, or tree;
- **Control**: represents basic programming logic, such as loops and conditions;
- **Operation**: are functionalities available in the mashup system that perform operations on the retrieved information. Examples include: filtering, merging, aggregating, arithmetic operations, and boolean logic operations;
- **Adaptation**: created based on the meta-information defined in wrappers, it represents the external resources accessed by the system.

To allow the reuse of existing mashups and build more sophisticated compositions, we also defined a component called **Reuse**. This component represents the compositions available in the mashup repository. Through the integration of this component with the aforementioned four, developers can extend a mashup created by some other developer.

D. Mashup System Prototype

Based on the presented architecture, we developed a network management mashup system prototype. This prototype focuses on enabling network managers to create their own management tools, even if they possess little to no programming experience. Our goals with this prototype are to evaluate the core concepts of our proposed architecture and to evaluate whether such architecture is useful in aiding a the development of a mashup system. Achieving these goals is one of the reasons we decided to develop a new mashup system instead of using an already existing tool. Furthermore, during our research, we could not find a mashup system suitable to the needs of network managers we perceive, such as compatibility with network management tools (*e.g.* MRTG) and protocols (*e.g.* SNMP), motivating us to build our own mashup system.

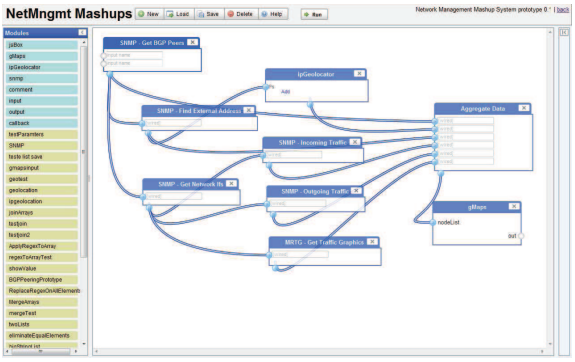

Fig. 3. Developer Environment

Our tool uses SOAP Web Services [12] as wrapper model, aiming to leverage the cross-plataform capabilities and the widespread standardization of this technology. As proposed by the architecture, these services are invoked by the composer, which is a server-side, JSP based engine. This engine comunicates with the presentation tier development environment, which is an AJAX based web system created with the WireIt library[4]. The development environment, as presented in Figure 3 presents a drag-and-drop interface for mashup development, where a user connect boxes with wires to represent the data flow. Each box represents a component, having a suitable number of inputs and, in this initial version, one single output. The mashups can also be executed in this environment. We also developed a set of components to this system, such as wrappers, operators and visual components, in order to create management tools. These components, as well as their use in the creation of a network mashup employing our prototype, are detailed in the next section.

[4]http://javascript.neyric.com/wireit/

On the next section, we also evaluate the proposed architecture and the prototype developed through the creation of a mashup whose specific goal is to integrate BGP routing information. The BGP peering issue addressed will be described and then we discuss the developed mashup.

IV. Case Study

Based on the architecture proposed in section 3, and using the developed prototype, we created a mashup that addresses BGP peering issues. Such issues are not easily tackled by administrators when using traditional network management tools. Our proposal is to employ Web 2.0 technologies in these problems to know if they present advantages over traditional management solutions. In this section we present the case study of managing BGP peering, discuss about the difficulties in doing so through conventional tools and present our mashup based solution.

A. BGP Peering

The Internet is an interconnection of independent administrative domains, most of them owned by Internet Service Providers (ISPs). Inside such domains, network administrators define policies (*e.g.,* for routing, traffic shaping, and traffic priorities) independently from other domains. For this reason, these domains are called Autonomous Systems (ASes). The connections between these ASes are maintained and controlled through the Border Gateway Protocol (BGP) [13]. When two ASes establish a connection, one announces routes to the other and vice-versa through BGP. These announcements (which are basically a set of IP address prefixes) indicate that one AS wants to receive traffic from the other. Two interconnected ASes are called *peers*, and the ASes interconnection is denominated *BGP peering* [14].

The main issues of BGP peering are related to unexpected behavior of the inter-AS relationship, for example, when a certain amount of expected traffic routed from/to a peer is suddenly replaced by a much higher or lower amount. This situation arises due to several reasons. For example, when a peer is not announcing its prefixes to the other peer, or when that peer is announcing an unconventional number of routes possibly breaking an SLA. This can happen when BGP routers are misconfigured or malfunctioning. Whichever the reason, the issue should be solved as soon as possible, given the large volume of traffic usually exchanged between ASes, which ends up leading to financial consequences too. In order to properly monitor the status of BGP peering, a network operator, located inside an AS, needs to have access, at least, to the following information:

- Local and remote BGP peers: a local BGP router is connected to remote BGP routers that belong to other ASes (*i.e.*, remote peers). Information about local and remote peers can include unique identifier (Autonomous System Number - ASN), ISP owner, and location of BGP routers;
- Number of announced routes: the number of routes announced by the local peer to remote ones, as well as the number of route announcements received from the remote peers to the local one. Usually, this number is defined by AS administrators when they establish SLAs between their ASes;
- Traffic exchanged: the amount of traffic exchanged between ASes. There is a financial cost associated to this exchange, which is also usually defined in SLAs. Network managers may define policies and strategies to split traffic between different peers in order to optimize costs.

The BGP-4 MIB [13] module, standardized by the IETF, provides both peer information and routing information via the Simple Network Management Protocol (SNMP). However, the associated analysis is not trivial. For example, in order to discover the IP address of remote ASes routers, the information retrieved from the local router's BGP-4 MIB module must be translated accessing complementary information from another MIB module: the MIB-II module. Also, to find the number of routes announced by each remote peer, the whole BGP-4 routing table must be retrieved from the local router via SNMP. That is so because this information (*i.e.*, number of routes announced per peer) is not supported in the BGP-4 MIB module [14] but can be calculated from the BGP-4 routing table. The task of discovering the amount of exchanged traffic is even more complex. This is done by checking which local network interfaces are connected to remote peers, and accessing the objects If.InOctects and If.OutOctects of the MIB-II module for each of these interfaces. This is insufficient for richer traffic analysis because these two objects do not provide per se, for example, temporal traffic averages, thus forcing the management station to pool the local BGP router to calculate the temporal traffic behavior in each interface. This is in fact the technique usually employed by traffic monitoring tools such as MRTG. However, the integration of such tools with information retrieved from the BGP-4 MIB module is difficult to be done. All this process is normally done manually by the network administrator, and at most through script programming.

B. Created Mashup

We developed the mashup to enable network administrators to manage the BGP peering issue previously presented. The final mashup aggregates routing information from local BGP routers and their remote peers on a map-based Web page. Map mashups are currently the most widely used mashup category on the Web [15] due to the increasing popularity of map APIs such as Google Maps[1], Yahoo Maps [2], and Microsoft's Bing Maps [3]. Network maps, on their turn, are classical tools used for the visualization of management topologies [16].

To build the mashup, the following components have been employed:

[1] http://code.google.com/apis/maps/
[2] http://maps.yahoo.com/
[3] http://www.microsoft.com/virtualearth/

- **SNMP**: an Adaptation component which allows the access to SNMP managed devices through a SNMP wrapper. Supports the protocol main operations (*e.g.*, GET, GET-NEXT, WALK), working as a gateway. Receives the target device IP, community string, operation to be realized and the accessed object OID as inputs. Outputs the operation results, received through SNMP;
- **Geolocation**: also an Adaptation component which interacts with *hostip.info* IP geolocation service. Receives an IP address as input and outputs the approximate geographic coordinates of the IP (*i.e.*, latitude and longitude); Its precision is limited, as are most, if not all, IP geolocation tools. This limitation is inherited by the created mashup, which depends on the service to position the routers in the map. Further discussion about IP geolocation precision is out of the scope of the present work;
- **Image extraction**: Adaptation component that extracts images from Web front-ends of external tools. It was used to pull graphics from MRTG, responsible for traffic statistics, and BGPe, responsible for announced routes statistics. This tools can be executed on the intranet or on the Internet, as the wrapper only requires a valid URL for them;
- **Graph**: a Reuse component (created using operation components) that aggregates information into a graph that represents the connections between a BGP router (*i.e.*, a root node) and its peers (*i.e.*, leaf nodes). Its output is a JSON string [17] that represents the graph;
- **Map**: a Visual component responsible for building the map. It is based on the information generated by the Graph component to draw the network topology, and display the retrieved statistics from the image component. The output is an HTML/JavaScript Web page that is rendered by users' Web browsers. For this work, we used the Google Maps API to create the maps.

In Figure 4, the design of the developed mashup is presented using the Business Process Modelling Notation (BPMN). The developer firstly inform the local BGP router address, its SNMP community string, and the MRTG and BGPe URLs on the developement environment. Based on that information, one SNMP wrapper retrieves the IPs of local BGP peers using the BGP-4 MIB module, and a second SNMP wrapper retrieves the IPs of external BGP peers. Having the set of external IPs, and using the MIB-2 module, the interfaces connected to external routers are retrieved from the local BGP router. For such interface, a SNMP request is done to retrieve the amount of traffic exchanged, and the correct statistics graphics from MRTG and BGPe. All IP addresses are so geolocated , and the graph operator assembles the representation of the network and feeds the map. Finally, The map wrapper creates the visualization of the mashup by placing visual elements representing the routers and connections on the map and annotating them with the relevant information.

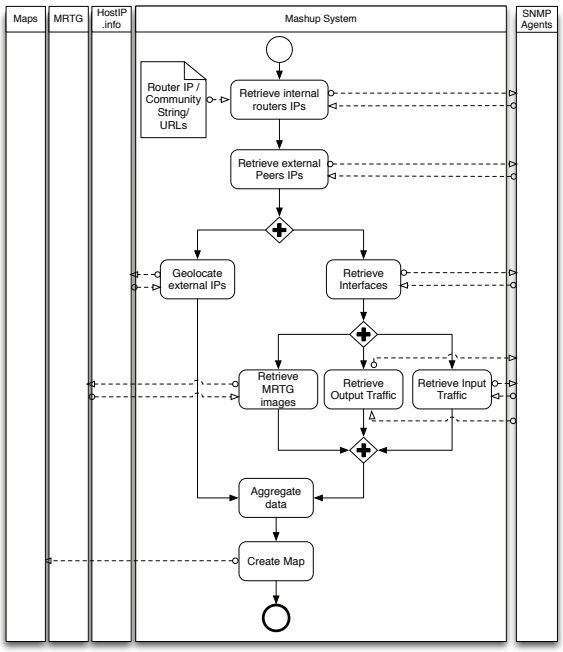

Fig. 4. Design of BGP router information visualization mashup

The wrappers (represented visually as adaptation components) have been developed as standalone applications. Thus, they can also be used by other composition systems that use a SOAP-based wrapper model or even a Java-based one, since the implementations under the SOAP interface were realized in Java. They have been designed with access interfaces that allow their internal implementation to be modified without affecting the external composition. For example, since the IP geolocation services are not accurate, the geo wrapper could be replaced by another one that access a better service.

The Runtime Environment, showed on Figure 5, presents the maps built to the user as Web pages, where the peers are represented as markers and the connections as lines linking those markers. Clicking on a visual element opens up a window where the user can access the pertinent information of the selected element, such as ASes involved in a connection, total traffic, traffic analysis graphics, and announced routes graphics. The user can also control the zoom level of the map according to his/her needs.

The information aggregated on the mashups allows an administrator to directly visualize the active BGP connections of a given set of managed ASes. Proceeding through a few intuitive interaction steps, mainly mouse clicks on elements representing BGP routers or connections, allows the manager to observe more detailed information as well as traffic and routes statistics. It is possible to insert more than one BGP router in one mashup, save it, and load the mashups again. The page itself is not saved, as it is dynamically created. Instead,

a mashup meta-description is persisted. When a mashup is loaded, this description is recovered and the composition is re-executed based on it, rebuilding the mashup with updated information. The mashup can also be programmed to update its information at runtime.

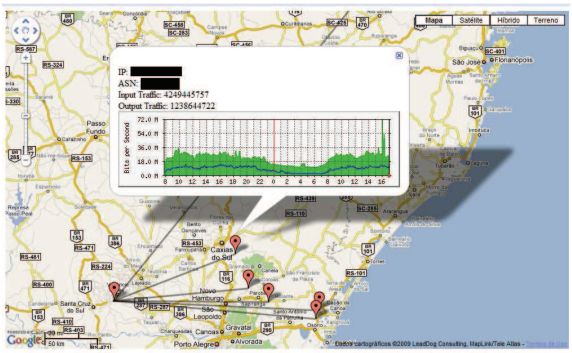

Fig. 5. *Runtime Environment* (BGP peering Mashup)

Originally, gathering BGP peering information requires the use of several tools with distinct interfaces. Besides, the integration itself is a complex process, which requires sophisticated scripts, handling large amounts of data, and effort to develop a suitable presentation to the results obtained. Our prototype reduces the interaction to the use of two user interfaces that do not require technical skills to be manipulated, keeping the complex technicalities of the integration transparent to the end user. In the development environment, the manager sets well known parameters (*e.g.,* IP addresses, URLs and SNMP community strings) so that the integration can be performed automatically by the underlying engine. The runtime environment exhibits the results of such integration into a map-based dynamic Web page. The popularity of map APIs and map-based Web pages is a strong indicative of their good usability, which the prototype inherits. The substantial decrease in the number of interaction steps and interfaces involved also contributes to the overall usability improvement of the prototype.

Having the discussion of prototype been presented, the next session will draw the conclusions obtained through its development and testing. After the presentation of the conclusions of this paper, we define our future work.

V. Conclusions and Future Work

The improvements in usability and agility provided by our prototype lead to the conclusion that the usage of mashups is both feasible and potentially advantageous to the network management field. The ever increasing growing of networks brings along more complexity to management, increasing the need for technologies that improve the network management process [16]. By enabling network managers to create their own management tools, a mashup-based approach can provide such improvement. It opens up the possibility to deal with situational issues that otherwise would be highly costly to address, such as the BGP peering issue presented. Verifying such possibilities provided by a mashup-based approach to network management is one of the main contributions of this work.

The development of the BGP peering mashup allowed us to observe that even exclusively network management information can be integrated with heterogeneous data through dynamic compositions. This can be useful for current networks, which usually employ different kinds of tools/services (*e.g.*, MRTG, Cisco Netflow, Looking Glass) in their infrastructures. We believe that with other wrappers (network management exclusively, or not), several other different management scenarios could be addressed. For example, integrating botnet related information to help network administrators to mitigate their attacks, employing tools such as online sandboxes.

The mashup-based approach is potentially better than traditional network management tools. It allows the creation of more meaningful information for the network administrators. The administrators don't need to spend time learning a specific programming language to create scripts. Situational changes in the management logic can be applied, tested, and deployed dynamically. Since mashups are essentially compositions, they can be shared with other administrators to be edited, and extended in a collaborative fashion. Due the decoupling of the building boxes of a mashup, its possible to spread them between different administrative domains, making possible a distributed management.

As future research, we aim at creating other wrappers, allowing our prototype be employed in different management situations. We will also investigate how to enable the compositions be run as daemons. They should be able to execute actions (to be represented as a new component), allowing not only the monitoring management, but also configuration management.

References

[1] J. Soldatos and D. Alexopoulos, "Web services-based network management: approaches and the wsnet system," *Int. J. Netw. Manag.*, vol. 17, no. 1, pp. 33–50, 2007.

[2] L. Z. Granville, D. M. da Rosa, A. Panisson, C. Melchiors, M. J. B. Almeida, and L. M. R. Tarouco, "Managing computer networks using peer-to-peer technologies," *Communications Magazine, IEEE*, vol. 43, no. 10, pp. 62–68, 2005. [Online]. Available: http://dx.doi.org/10.1109/MCOM.2005.1522126

[3] T. O'Reilly, "What is web 2.0," 2005, http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html.

[4] D. Merril, "Mashups: The new breed of web app - an introduction to mashups." 2003, http://www.ibm.com/developerworks/web/library/x-mashups.html.

[5] X. Liu, Y. Hui, W. Sun, and H. Liang, "Towards service composition based on mashup," in *Services, 2007 IEEE Congress on*, 2007, pp. 332–339. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4278815

[6] R. Ennals and D. Gay, "User-friendly functional programming for web mashups," in *ICFP '07: Proceedings of the 2007 ACM SIGPLAN international conference on Functional programming*. New York, NY, USA: ACM, 2007, pp. 223–234.

[7] J. J. Garret, "Ajax: A new approach to web applications," 2005, http://www.adaptivepath.com/ideas/essays/archives/000385.php.

[8] N. Banerjee, K. Dasgupta, and S. Mukherjea, “Providing middleware support for the control and co-ordination of telecom mashups,” in *MNCNA*. ACM, 2007, p. 11. [Online]. Available: http://doi.acm.org/10.1145/1376878.1376889

[9] J. Yu, B. Benatallah, F. Casati, and F. Daniel, “Understanding mashup development,” *IEEE Internet Computing*, vol. 12, no. 5, pp. 44–52, 2008.

[10] V. Hoyer, K. Stanoevska-Slabeva, T. Janner, and C. Schroth, “Enterprise mashups: Design principles towards the long tail of user needs,” in *IEEE SCC*. IEEE Computer Society, 2008, pp. 601–602. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/SCC.2008.88

[11] W. W. Eckerson, “Three tier client/server architecture: Achieving scalability, performance, and efficiency in client server applications,” Open Information Systems, 1995.

[12] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, “Unraveling the Web Services Web - An Introdution to SOAP, WSDL,and UDDI,” *IEEE Internet Computing*, vol. 6, no. 2, pp. 86–93, 2002.

[13] Y. Rekhter and T. Li, “A border gateway protocol 4 (bgp-4),” United States, 1995.

[14] G. D. Battista, T. Refice, and M. Rimondini, “How to extract bgp peering information from the internet routing registry,” in *MineNet ’06: Proceedings of the 2006 SIGCOMM workshop on Mining network data*. New York, NY, USA: ACM, 2006, pp. 317–322.

[15] J. Wong and J. Hong, “What do we ”mashup” when we make mashups?” in *WEUSE ’08: Proceedings of the 4th international workshop on End-user software engineering*. New York, NY, USA: ACM, 2008, pp. 35–39.

[16] F. Kamoun, “Toward best maintenance practices in communications network management,” *Int. J. Netw. Manag.*, vol. 15, no. 5, pp. 321–334, 2005.

[17] D. Rubio, “An introduction to json,” 2007, http://dev2dev.bea.com/pub/a/2007/02/introduction-json.html.